AFIT/DS/ENG/94J-04



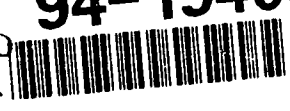
# EMBEDDED CHAOTIC TIME SERIES: APPLICATIONS IN PREDICTION AND SPATIO-TEMPORAL CLASSIFICATION

DISSERTATION

James R. Stright
Captain, USAF

AFIT/DS/ENG/94J-04





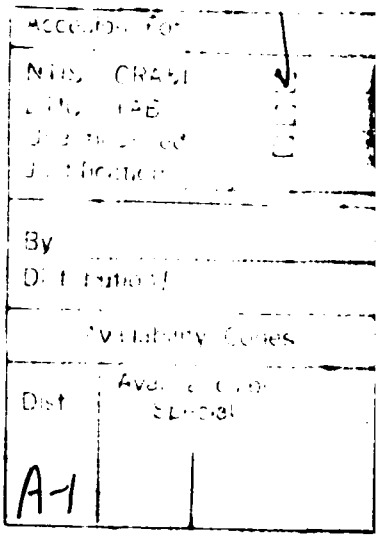
AFIT/DS/ENG/94J-04

# EMBEDDED CHAOTIC TIME SERIES: APPLICATIONS IN PREDICTION AND SPATIO-TEMPORAL CLASSIFICATION

DISSERTATION

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

James R. Stright, B.S., B.E.E., M.S.E.E.
Captain, USAF




June, 1994

AFIT/DS/ENG/94J-04

# EMBEDDED CHAOTIC TIME SERIES: APPLICATIONS IN PREDICTION AND SPATIO-TEMPORAL CLASSIFICATION

James R. Stright, B.S., B.E.E., M.S.E.E.

Captain, USAF

Approved:

| | |
|---|---|
| | 16 MAY 94 |
| Dennis W. Ruck | 16 MAY 1994 |
| Mark E. Oxley | 16 May 94 |
| Dennis Wunsch | 16 MAY 94 |

J. S. Przemieniecki 26 May 94

J. S. PRZEMIENIECKI
Institute Senior Dean and
Scientific Advisor

## *Acknowledgements*

I am deeply indebted to many people for inspiration, assistance, and guidance in this effort. Thanks first to my advisor, Dr. Steve Rogers, who whetted my interest in chaos and time series prediction some time ago and stuck around to keep if from drying. His breadth of experience, eye for good papers, and encouragement have been essential to my research. I am also especially grateful to Dr. Dennis Quinn, whose steady support and shared insights have been indispensable.

Thanks also to my other committee members, Dr. Mark Oxley, who introduced me to the mathematics underlying fractals, and Capt Dennis Ruck, whose knowledge of the computer jungles guided me safely past many snakes.

My fellow prognosticator Dale Nelson has steadfastly provided not only knowledge and encouragement but also a wealth of references, conference information and time series data.

The spatio-temporal pattern recognition work of my colleague Capt Ken Fielding figures prominently in this dissertation. Less obvious but no less important were his countless contributions to my computer literacy. Capt Tom Burns provided computer literacy, shared interests, *and* his job in Florida. Thanks, Ken and Tom.

The folks from Neuronetics Limited of Dayton, OH, shared a number of insights into financial markets, and provided me several financial time series.

Too many AFIT faculty members have helped me to list all of them. Dr. Peter Maybeck has been an inspiration and a treasured mentor. Dr. David Barr and Dr. William Wiesel have shared with me their knowledge of, and enthusiasm for, chaotic phenomena.

Last but not least, to my wife, Linda, and son, Travis, for putting up with my absence even when I'm with them: gigathanks.

James R. Stright

## *Table of Contents*

## *List of Figures*

## *List of Tables*

## *List of Symbols*

## *Abstract*

The Deterministic Versus Stochastic algorithm developed by Martin Casdagli is modified to produce two new prediction methodologies, each of which selectively uses embedding space nearest neighbors. Neighbors which are considered prediction-relevant are retained for local linear prediction (which is shown a reasonable alternative to local nonlinear prediction), while those which are considered likely to represent noise are ignored. The new algorithms may in this sense be considered to employ embedding space filtrations of the time series. For many time series, it is shown rather easy to improve on unfiltered local linear prediction with one or both of the new algorithms. For other time series, prediction improvement is more difficult. It is suggested that prediction improvement difficulty is indicative of stochastic data, independently of the direct results of the Deterministic Versus Stochastic algorithm. The theory of embedded time series is also shown capable of determining a reasonable length of test sequence sufficient for accurate classification of moving objects. Sequentially recorded feature vectors of a moving object form a training trajectory in feature space. Each of the sequences of feature vector components is a time series, and under certain conditions, each of these time series will have approximately the same fractal dimension. The embedding theorem may be applied to this fractal dimension to establish a number of observations sufficient to determine the feature space trajectory of the object. It is argued that this number is a reasonable test sequence length for use in object classification. Experiments with data corresponding to five military vehicles (observed following a projected Lorenz trajectory on a viewing sphere) show that this length is indeed adequate.

# EMBEDDED CHAOTIC TIME SERIES: APPLICATIONS IN PREDICTION AND SPATIO-TEMPORAL CLASSIFICATION

## *I. Introduction*

### *1.1 Background*

It is often easier to measure and record data than it is to explain how the data was generated. It is one thing, for example, to record the inflight positions of an enemy aircraft but quite another to set forth equations which will allow accurate prediction of its future behavior. Throughout this document, a time series will refer to a discrete time-ordered set of real numbers corresponding to a single component of a dynamical system (although less restrictive definitions are common (14:3)). In the case of a flying aircraft, a time series might be recorded for the measured longitudinal position of the aircraft; another time series might represent its altitude.

Some time series are predictable with much greater accuracy than others. The position of a drone aircraft with locked controls is far easier to predict than the position of a piloted aircraft conducting evasive maneuvers. Certain factors lend a degree of unpredictability, however, to both situations. In particular, unknown atmospheric conditions (such as winds aloft) limit the long term predictability of both. Other factors sharply limit even the short term predictability of the piloted aircraft (for instance, the intellect and will to survive of the pilot). Loosely, a chaotic time series refers to a time series generated by a dynamical system having a large number of poorly understood controlling influences. Although chaotic is a relative term, time series generated by the piloted aircraft are more likely to be referred to as chaotic than are those generated by the drone.

More precise definitions of chaotic dynamical systems suggest the often intricate, fractal structure of their phase space portraits (25:341). It wasn't until the early 1960's that the existence of what is now called a chaotic dynamical system was conclusively demonstrated (21). As understanding of this phenomenon has increased, it has been realized that a sensitivity to initial conditions is inherent in chaos. Even given the equations governing a chaotic system, sensitivity to initial conditions destroys long term predictability in the sense that, given an *approximation* to an object's phase space location, the divergence of nearby trajectories renders worthless attempts to infer future nearness from current nearness.

Nevertheless, it is possible to accurately predict chaotic time series, sometimes remarkably far into the future. Probably the best current understanding of how this is possible has its foundations in what Tim Sauers has called the theory of embedology (33). This will be discussed more completely in Chapter II, but roughly what it involves is reducing the initial complexity of a dynamical system to a few essential degrees of freedom and embedding the given time series in a space whose dimension is only about twice that number of degrees. An important theorem published by Takens (37) in 1981 (its generalization by Sauers will be called the embedding theorem) guarantees that the resulting embedding is a faithful reproduction of the original dynamics. Knowledge of the behavior of portions of the embedded time series near the portion corresponding to its end value may then be exploited to form predictions. Farmer took this approach in 1987 in formulating what he called a local linear prediction technique (8).

## *1.2 Problem Statement*

Greater prediction accuracy is always desirable. The local linear prediction technique will be enhanced by excluding detracting global influences from the local prediction region. A central tenet of the technique, the embedding theorem, will also be shown to extend to spatiotemporal pattern recognition.

### *1.3 Scope*

Farmer's basic local linear approximation methodology has since been refined by a number of researchers; for example, (4, 33). Most refinements, however, retain the basic technique of using a *globally* applicable embedding dimension and *globally* applicable number of nearest neighbors to predict from any given point in a time series. It might be imagined that applying a number of nearest neighbors appropriate to prediction from a single closest point in embedding space (an extremely local approach) might offer prediction improvement. This was found not necessarily the case. Improved predictability *can* often be realized by pruning from a globally determined number of nearest neighbors, those neighbors which are least relevant to the local behavior of the dynamical system, provided the system is not extremely stochastic. A pruning technique is introduced which eliminates from a globally ascertained number of nearest neighbors, those neighbors which lie farthest from the regression hyperplane that they determine. It is further shown that pruning from sets of neighbors determined by different choices of delay interval, those which do not overlap in time, can also enhance prediction accuracy. These contentions are supported by examining the performances of both new algorithms against pure local linear prediction for six time series. Ten predictions are provided for each of the six time series. Both techniques introduced here enhance predictability by reducing noise, in that they provide the most striking benefits for time series usually considered noisy, such as experimental measurements and financial data. These approaches are considered in Chapter III.

The existence of fractal qualities in time series may be exploited not only for prediction but also for classification. Considerable recent research focuses on the problem of identifying various objects using not only the spatial content of observations, but also information imparted by the evolution in time of those observations (34, 20, 3, 12). Training sequences of feature vectors are obtained from numerous sequentially obtained views of the objects. Test sequences are then compared to the training data, but thus far, little theoretical justification has been provided for an appropriate test sequence length. A new view of training sequences is here provided. Training sequences can be interpreted as projected solution curves

of dynamical systems. It is shown in Chapter IV that the embedding theorem, applied to the fractal dimensions of the solution curves, yields a required length of test sequence which experimentation reveals quite adequate for a classification task.

### *1.4 Outline*

The following chapter presents some of the fundamental theory underlying prediction of chaotic time series. Chapter III develops two modifications of the local linear prediction method introduced by Farmer in 1987 (8). Chapter IV presents an application of the embedding theorem to the problem of moving object recognition. Chapter V presents some conclusions.

# *II. Background*

This chapter presents some of the fundamental theory underlying prediction of chaotic time series. The first section provides some concepts from dynamical systems theory. The second section provides an overview of a commonly applied technique for extracting the correlation dimension from time series data. A computer program which implements this algorithm is included in Appendix C. Section three shows how knowledge of correlation dimension can be applied to find an appropriate embedding dimension for a time series.

## *2.1 Basics of Dynamical Systems Theory*

Throughout this document, a time series will refer to a discrete time-ordered set of real numbers corresponding to a single component of a dynamical system. In a broad sense, a *dynamical system* is simply a map $f : X \rightarrow X$ of a metric space $(X, d)$ (1:60) (i.e., set $X$ with metric $d$) into itself. The *orbit* of a point $x$ in $X$ is the sequence of points $\{x, f(x), f \circ f(x), f \circ f \circ f(x), \ldots\}$ (2:134). For time series extraction, attention is restricted to the particular metric spaces $\Re^n, n \geq 1$, or subsets of them, with the metric specified in context.

Consider the metric space $X = [0, 1]$ with the Euclidean metric; that is, the closed unit interval where the distance between two points is the absolute value of their difference. The identity map on $X$ is a dynamical system. So, too, is the map $f$ which takes each element $x$ of $X$ to $x/2$. The orbit of the point 1 under $f$ is the sequence $\{1, 1/2, 1/4, 1/8, \ldots\}$.

A more interesting type of dynamical system can be defined on a subset $\mathcal{C}$ of $[0, 1]$ called the (classical) Cantor set (1:180). The Cantor set can be formed by iteratively deleting open middle intervals from $[0, 1]$. Figure 1 shows the first few steps in its generation. Letting $B_0 = [0, 1]$, $B_1 = [0, 1/3] \cup [2/3, 1]$, $B_2 = [0, 1/9] \cup [2/9, 1/3] \cup [2/3, 7/9] \cup [8/9, 1]$, $\ldots$, $\mathcal{C}$ is defined as $\cap_{i=0}^{\infty} B_i$. However, it is possible to view the Cantor set as the unique "fixed point," or "attractor," of an "iterated function system" (2:82). To this end, define two "contraction mappings" $w_1$ and $w_2$ from $\Re$ into $\Re$ by $w_1(x) = x/3$ and $w_2(x) = x/3 + 2/3$.

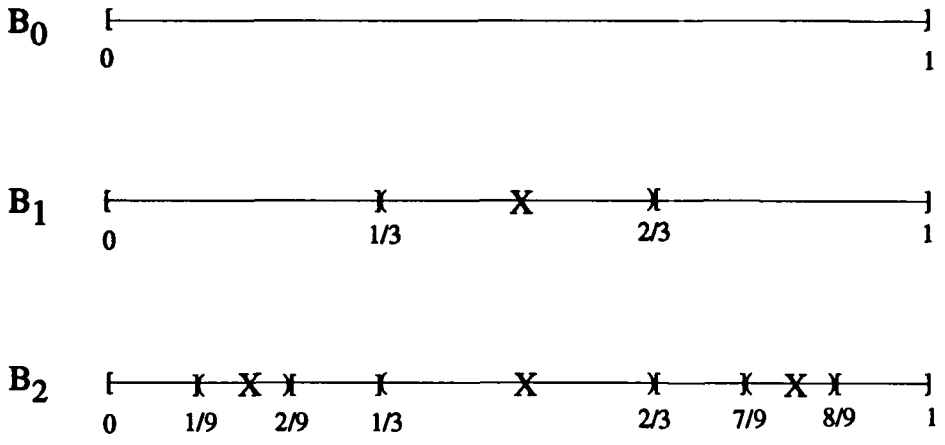



Figure 1. Building the Cantor set

An intuition of the reason for the name "contraction mappings" can be gained by noticing that these functions shrink any interval on which they operate (by a factor of 1/3). Together with the space $\Re$ on which they operate, $w_1$ and $w_2$ form an iterated function system. The term "iterated function system" derives from the fact that these functions are understood to operate jointly and repeatedly on $\Re$, or a subset of $\Re$. Consider the actions of these functions on the closed unit interval.

At the first iteration, $w_1$ takes $[0,1]$ to $[0,1/3]$, and $w_2$ takes $[0,1]$ to $[2/3,1]$. Thus the set $B_1$ results from one application of $w_1$ and $w_2$. The second iteration consists of applying $w_1$ and $w_2$ to $B_1$. The resulting set is $B_2$. With very few further iterations, it should be clear to the reader that the "infinite iteration" of this iterated function system is the Cantor set. In fact, Barnsley provides a theorem which assures that $\mathcal{C}$ will result from applying $w_1$ and $w_2$ to any nonempty compact subset of $\Re$ (2:82). It is in this sense that $\mathcal{C}$ is referred to as the attractor of this iterated function system.

The Cantor set is perhaps the quintessential example of a fractal set. A rough "mind's eye" image of it is available in Figure 2; it actually consists of uncountably many points but contains no intervals (2:134) (6:39). Like many fractals, it is self-identical across scale; for


Figure 2. Low Resolution Cantor Set C

example, if the portion of $C$ lying in the interval $[0, 1/9]$ is expanded by a factor of 9, all of $C$ is retrieved.

A dynamical system $S : \mathcal{C} \rightarrow \mathcal{C}$ can be defined by taking the image $S(a)$ of any number $a$ in $\mathcal{C}$ to be the preimage of the only map $w_1 : \mathcal{C} \rightarrow \mathcal{C}$ or $w_2 : \mathcal{C} \rightarrow \mathcal{C}$ whose range contains $a$. This type of dynamical system is called a *shift* dynamical system (2:144). Appendix A contains an elaboration of iterated function systems, shift dynamical systems, and related concepts.

The word *dynamical* in the preceding definition of dynamical system comes from the repeated discrete actions of a function on a set — the dynamics implied is the discrete evolution of points under iterations of the defining map. Another definition allows for the continuous motion of points in a set (18:160). Accordingly, a *dynamical system* is a continuously differentiable map $\phi : \Re \times \mathcal{S} \rightarrow \mathcal{S}$ where $\mathcal{S}$ is an open subset of $\Re^n, n \geq 1$, and letting $\phi(x,t) = \phi_t(x)$, the map $\phi_t : \mathcal{S} \rightarrow \mathcal{S}$ satisfies (1) the map $\phi_0 : \mathcal{S} \rightarrow \mathcal{S}$ is the identity; and (2) the composition $\phi_t \circ \phi_s = \phi_{t+s}$ for each $t, s$ in $\Re$.

The parameter $t$ is usually associated with time; thus the magnitude of $t_2$ (relative to some previous time $t_1$) gives an indication of how long a point $x$ has wandered about in $\mathcal{S}$;

and $\phi_{t_2}(x)$ gives the point in $\mathcal{S}$ to which it has arrived at time $t_2$. Property (2) demands that a terminal location be independent of choice of initial time.

With this definition, every dynamical system gives rise in a natural way to a differential equation (18:160). Conversely, the fundamental theorem of differential equations shows that every differential equation gives rise to a dynamical system. Consider, for example, the differential equation

$$\mathbf{x}' = \mathbf{A}\mathbf{x} \tag{1}$$

where

$$\mathbf{x}' = \begin{bmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & -1/2 \end{bmatrix}, \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

For any initial condition $\mathbf{x}_0 = (x_1(0), x_2(0))^\mathsf{T}$, this equation has the unique solution

$$\phi_t(\mathbf{x}) = (x_1(0)e^{2t}, x_2(0)e^{-t/2})$$

Given any point $(x_1, x_2)$ in $\Re^2$, therefore, there is a unique solution curve $\phi_t(\mathbf{x})$ passing through that point. Such solution curves are called *trajectories*. A trajectory may be thought of as the path taken by a particle placed at the point $(x_1, x_2)$ subject to a set of forces represented by Equation 1. The set of all points in the plane are acted on by the map $\phi_t, t \in \Re$ to determine a family of trajectories, which are collectively called the *flow* on $\Re^2$ determined by Equation 1. A qualitative representation of the flow determined by Equation 1 is given in Figure 3.

In dynamical systems theory, the space $\Re^2$ of Figure 3 is called *phase space* (25:438). The term *state space* is sometimes used instead (18:22). The current *state* of a particle — that is, its location $(x_1, x_2)$ — is all the information needed to characterize for all time the motion of the particle under the influence of the forces represented by Equation 1.

This last fact is a consequence of the fundamental theorem of (ordinary) differential equations and warrants emphasis. One version of this theorem may be stated as follows (18:162). Let $W$ be an open subset of $\Re^n$, $\mathbf{f} : W \to \Re^n$ a continuously differentiable

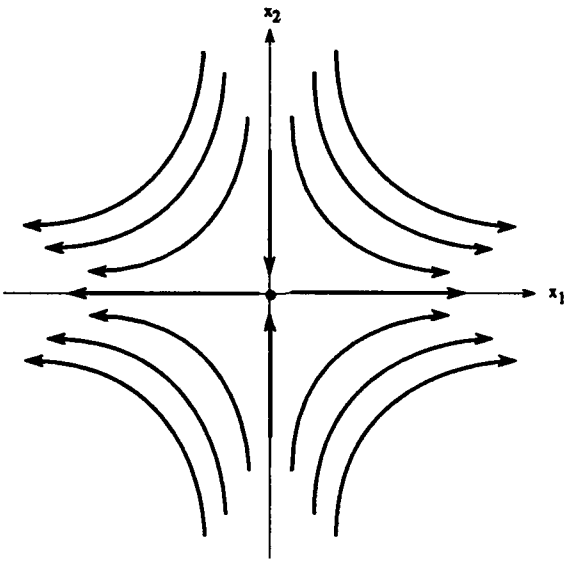



Figure 3. Some Solution Curves to $\mathbf{x}' = \mathbf{A}\mathbf{x}$

map, and $\mathbf{x}_0 \in W$. Then there is some $a > 0$ and a unique solution $\mathbf{x} : (-a, a) \rightarrow W$ of the differential equation $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ satisfying the initial condition $\mathbf{x}(0) = \mathbf{x}_0$.

This theorem assures both the existence and uniqueness of solutions to a large class of differential equations. As an immediate consequence of uniqueness, two solution curves of $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ cannot cross. Furthermore, a solution curve cannot cross itself (18:168). Thus solution curve(s) cannot evolve through $\Re^n$ as depicted in Figure 4. A solution curve which revisits the same point in $\Re^n$ must close up as shown in Figure 5.

A *dissipative* dynamical system is one which contracts phase space volumes. Suppose, for example, that all points in the phase space of Figure 5 were on trajectories that converged to the closed curve depicted. Then any small volume element in the phase space would ultimately be contracted onto the closed curve. Since the closed curve has volume zero, the volume element was contracted by the dynamical system. Thus the dynamical system would be dissipative. On the other hand, if trajectories far outside the closed curve diverged away from it, then some volume elements would expand and the dynamical system would not be dissipative.

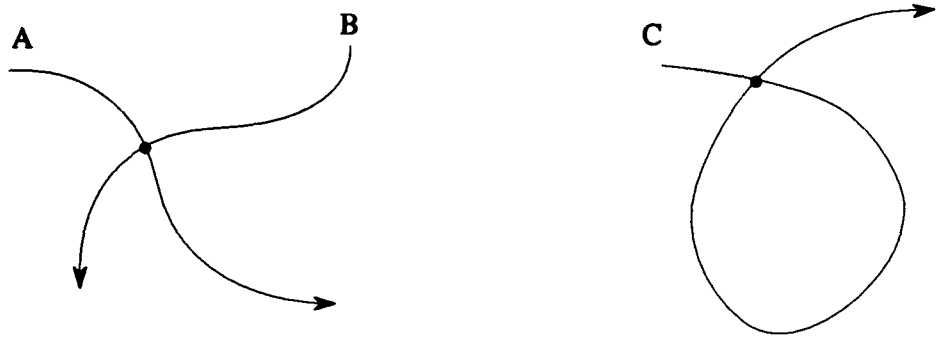



Figure 4. Impossible Trajectories

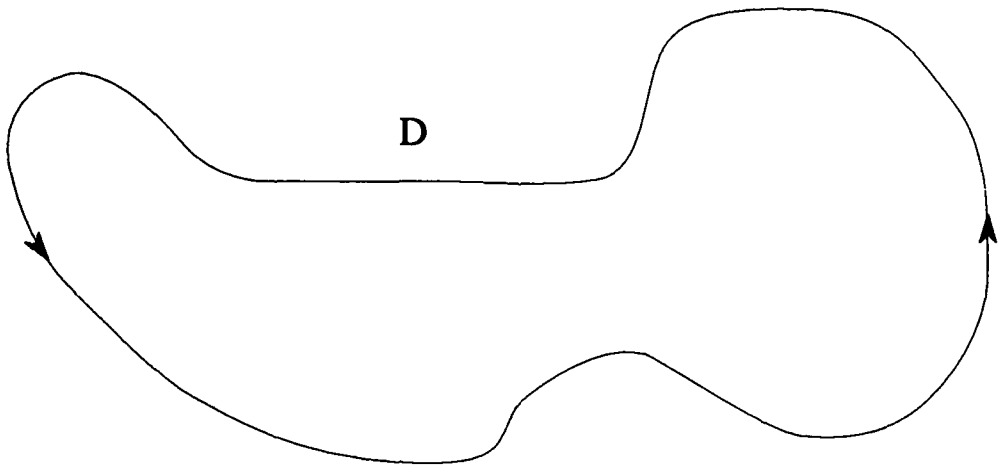



Figure 5. A Possible Trajectory

A *chaotic* dynamical system is a dissipative dynamical system which has one or more positive Lyapunov exponents (40). Wolf et al provide a good explanation of Lyapunov exponents and their relationship to time series (42). This reference also provides source code for a program which finds positive Lyapunov exponents. All numerical methods for determining Lyapunov exponents, however, involve rather subjective choices of parameters, and there is often disagreement over whether any particular time series is or is not chaotic.

Essentially, the existence of at least one positive Lyapunov exponent assures that a chaotic dynamical system stretches small volume elements in one or more directions; but since the system is dissipative, the volume elements contract to volume zero. For example, a small cube in $\Re^3$ might be transformed into a long curve in $\Re^3$. Perhaps the most celebrated chaotic dynamical system is given by the Lorenz equations (21:135):

$$\begin{aligned} x' &= -\sigma x + \sigma y \\ y' &= -xz + rx - y \\ z' &= xy - bz \end{aligned}$$

A solution curve for this system of equations (corresponding to the initial condition $x = y = z = 5$, and with $\sigma = 10, r = 28$, and $b = 8/3$) winds through a bounded region of $\Re^3$, without ever intersecting itself (16:29). Its projection into the $(x, y)$-plane is approximated in Figure 6. This figure also provides a "mind's eye" image of the projection of the "limit set" in $\Re^3$ toward which trajectories from almost all initial points converge. This limit set is called the *strange attractor* associated with the Lorenz equations, or sometimes simply the Lorenz attractor. The attractor of a dynamical system is of considerable interest because it represents the long term behavior of the system.

### 2.2 *Dimensionality of Chaotic Trajectories*

Often a strange attractor will have associated with it a fractal quality such as exhibited by the Cantor set. For example, (apparent) pairs of inner loops of the strange attractor

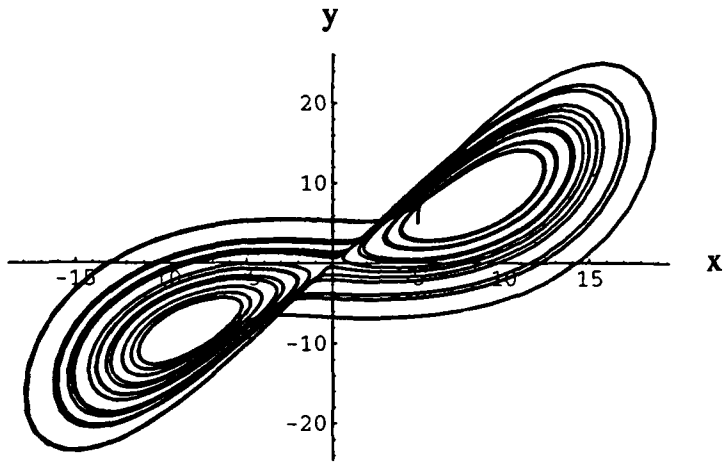



Figure 6. A Lorenz Trajectory

approximated by Figure 6 appear to be nestings of pairs farther outside. It turns out that a single noninteger number can characterize the self-similarity of the Lorenz attractor. Other attractors may fail to display a fractal quality (7:625).

The quantity used to characterize the self-similarity of trajectories such as those which approximate the Lorenz attractor is called *fractal dimension.* Actually *fractal dimension* is something of a catchall phrase used to describe a number of such quantifications, although some authors do rigorously define the term (usually in a manner consistent with what is descriptively known as the *box counting dimension* (2:174)).

Theiler provides a geometric intuition of fractal dimension as a generalization of the more commonly understood notion of dimension of geometric objects (39:1059). Dimension can be interpreted as an exponent that expresses the scaling of an object's bulk with its size:

$$\text{bulk} \sim \text{size}^{\text{dimension}} \tag{2}$$

The "bulk" in this formulation may correspond to a mass, a volume, or some measure of information content, and "size" is a linear distance. The volume (bulk) of an object in $\Re^3$ such as a sphere, for example, scales cubically with its diameter (size), so the object is three

dimensional. Accordingly, the various definitions of fractal dimension are usually cast as equations of the form

$$\text{dimension} = \lim_{\text{size}\to 0} \frac{\log \text{bulk}}{\log \text{size}}$$

where the limit of small size is taken to ensure that an object's fractal dimension will not change if it is linearly transformed (for instance, by a rotation). This small-size limit also makes dimension a local quantity so that a global definition of fractal dimension requires some kind of averaging.

This global averaging is achieved automatically by the definition of *capacity* or box counting dimension (25:330) (2:176). Consider an attractor $A$ in $\Re^m$. Cover $\Re^m$ by closed just-touching hypercubes of side length $Cr^n$ where $C > 0$ and $0 < r < 1$ are fixed real numbers; see Figure 7, where $m = 2, C = 1$, and $r = 1/2$. Let $\mathcal{N}_n(A)$ denote the number of

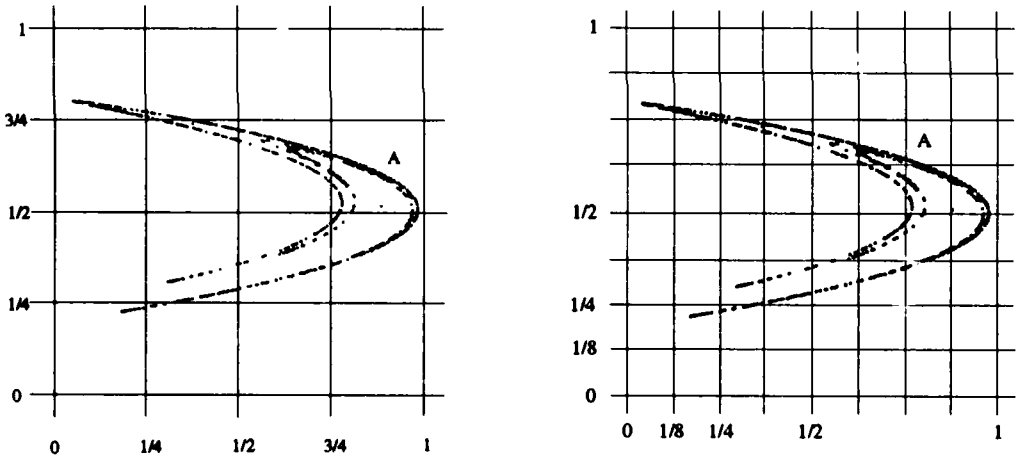



Figure 7. Finding Capacity Dimension; $\mathcal{N}_2(A) = 10$ and $\mathcal{N}_3(A) = 23$

hypercubes of side length $Cr^n$ which intersect $A$. If the limit

$$D = \lim_{n\to\infty} \left\{ \frac{\ln(\mathcal{N}_n(A))}{\ln(1/Cr^n)} \right\}$$

exists, then $A$ has *capacity dimension* $D$.

A time series $x_1, x_2, x_3, \ldots$ consists of a sequence of real numbers, hence there is little a priori geometric intuition behind the notion of fractal dimension of a time series. Suppose, however, that the time series consists of samples equally spaced in time and is *delay coordinate embedded* into a space of dimension higher than one, in the following manner. Fix some small integer $k \geq 2$. Form from the time series a set of points $\mathbf{x}_i, i = 1, 2, \ldots$ in $\Re^k$ by taking successive $k$-tuples from the sequence $x_1, x_2, x_3, \ldots$. That is,

$$\begin{aligned} \mathbf{x}_1 &= (x_1, x_2, \ldots, x_k) \\ \mathbf{x}_2 &= (x_2, x_3, \ldots, x_{k+1}) \\ \mathbf{x}_3 &= (x_3, x_4, \ldots, x_{k+2}) \\ &\vdots \end{aligned}$$

Then the resulting set of points may indeed have a fractal structure in $\Re^k$, similar perhaps to the attractor $A$ depicted in Figure 7. The spaces $\Re^k$ are called *embedding* or *reconstruction* spaces.

Grassberger and Procaccia provided in 1983 an elegant algorithm for finding a fractal dimension of a time series using just such delay coordinate embeddings (17). They termed the dimension which results from their algorithm the *correlation exponent* (denoted $\nu$) for the time series, but the term *correlation dimension* is now commonly used. They demonstrated that correlation dimension is bounded above by information dimension (another commonly used measure of fractal qualities (38)) which in turn is bounded above by capacity dimension. For most attractors studied, these bounds are quite tight. As will be seen, great accuracy of fractal dimension estimation is not necessary for implementation of the embedding theorem, which is the foundation for the local linear prediction method. For this reason, the term fractal dimension will often be used to mean the approximately equal value of any of these quantities.

The Grassberger and Procaccia algorithm is easy to describe and to implement; Reference (36:3-7), for example, provides a succinct explanation. It involves examining, for each of a small set of embedding dimensions $k, k+1, k+2, \ldots, k+s$, the embedded time series

data to see if it exhibits an exponential scaling as in Equation 2. Here $k$ is taken slightly larger than the actual correlation dimension $\nu$; since $\nu$ is unknown, additional embedding dimensions are used to ensure that $\nu$ has indeed been exceeded, and that $k + s$ exceeds about $2\nu$. The condition indicative of the existence of a fractal structure, the linearity of log-log plots, is always displayed for values of embedding dimensions in excess of about $2\nu$ (and often for smaller values). In fact, the value of $\nu$ is the common slopes of these plots.

Using the embedding dimension $k$, the Grassberger and Procaccia method locates all contiguous $k$-tuples from the time series as points in $\Re^k$. A set of small numbers $l_i$ is chosen (about eight is adequate), and for each $i$ the number of pairs of contiguous $k$-tuples within Euclidean distance $l_i$ of each other is determined and denoted $C(l_i)$. Figure 8 illustrates a

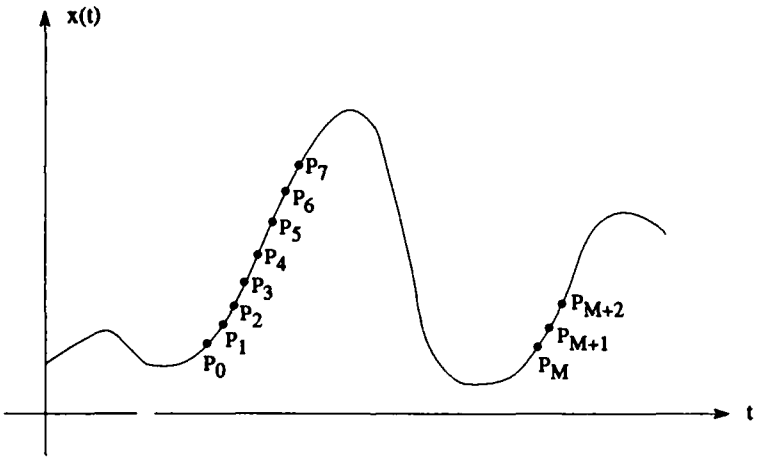



Figure 8. Counting Close $k$-tuples ($k = 3$)

case where the 3-tuple $(P_M, P_{M+1}, P_{M+2})$ is close to $(P_0, P_1, P_2)$ in Euclidean distance and would likely contribute to the count $C(l_i)$ for most $l_i$. On the other hand, $(P_5, P_6, P_7)$ is relatively far from $(P_0, P_1, P_2)$ and might not contribute to the count $C(l_i)$ for any of the selected $l_i$. The Grassberger algorithm can be implemented by taking the smallest $l_i$ and the leftmost 3-tuple $(P_0, P_1, P_2)$ and checking each 3-tuple to the right for its adherence to the $l_i$ distance criterion - first $(P_1, P_2, P_3)$, then $(P_2, P_3, P_4)$, continuing until all contiguous 3-tuples are compared with $(P_0, P_1, P_2)$. After the rightmost 3-tuple has been compared, the

process repeats beginning at the left with $(P_1, P_2, P_3)$. The $C(l_i)$ counter continues to grow as more pairs of 3-tuples are found to fall within the allowable $l_i$ distance. After the rightmost 3-tuple is compared with $(P_1, P_2, P_3)$, $(P_2, P_3, P_4)$ serves as the leftmost 3-tuple for the next iteration.

Continuing in like manner, leftmost finally becomes rightmost, and the counter $C(l_i)$ indicates the total number of contiguous 3-tuples which lie within distance $l_i$ of each other. The numbers $l_i$ and $C(l_i)$ are stored, another $l_j$ is selected, and the process begins anew with $(P_0, P_1, P_2)$ at the left. It terminates with $C(l_j)$ determined. All pairs $(l_n, C(l_n))$ are likewise determined and stored.

If the pairs $(\ln(l_i), \ln(C(l_i)))$ are plotted, and are found nearly collinear, then the slope of a line through these points provides a good approximation of the fractal dimension of the dynamical system which produced the time series. See Figure 36 and Figure 40 for examples of such plots; $L0 = 1$ in these plots.

A local linear prediction methodology will be described in the next section. This methodology is based on the existence of a unique finite fractal dimension of a solution trajectory, rather than on the chaos of the system which produced it. It might make more sense, therefore, to speak of *fractal* time series prediction rather than *chaotic* time series prediction. Following Farmer (8), however, the latter terminology is preserved.

### *2.3 Embedology and Prediction*

The process of successfully extracting a correlation dimension $\nu$ reveals that beyond a certain integer dimension $m$ for the delay coordinate embedding space, the calculated value of $\nu$ remains nearly constant. Putting the numerical process of dimensionality determination aside, there is (subject to certain mild technical conditions (33:13)) a theoretical value $M$ of embedding space dimension beyond which $\nu$ cannot change. This is because for values $m \geq M$, the delay coordinate embedding of the state space trajectory into reconstruction space $\Re^m$ is (in a probability-one sense) a smooth diffeomorphism, and correlation dimension is invariant under such mappings (32:178) (27:368). (A diffeomorphism is a differentiable mapping from

one open subset of a vector space to another with a differentiable inverse (18:242); to say that it is smooth means that its derivative is continuous (33:6).)

More to the point, Sauer showed that as long as $M > 2d$, where $d$ is the box counting dimension of the trajectory in state space $\Re^n$, the delay coordinate mapping from the trajectory into $\Re^M$ is one-to-one (32:178). This is the essence of what he called the fractal delay coordinate embedding theorem, which is a generalization of Takens' theorem. Although correlation dimension is always less than or equal to the box counting (aka fractal (2:176)) dimension, the difference between them is rarely larger than 0.05 (17:193) and the embedding theorem will often be invoked using box counting, fractal and correlation dimensions interchangeably.

The one-to-one property has important implications for prediction. The fundamental theorem of differential equations assures that a phase space solution trajectory is completely determined by any point on it (18:162). A delay coordinate embedding into reconstruction space $\Re^m$, $m \geq M$, shares this property. In particular, given any point $(y(t), y(t-\tau), \ldots, y(t-(m-1)\tau))$ on the reconstruction space trajectory, its image on the phase space trajectory has a unique succeeding point. The $y$-component of that succeeding point is $y(t+\tau)$; see Figure 9. If the point $(y(t), y(t-\tau), \ldots, y(t-(m-1)\tau))$ reappears on the reconstruction space trajectory at a time $t^\star > t$, so that $(y(t^\star), y(t^\star-\tau), \ldots, y(t^\star-(m-1)\tau) = (y(t), y(t-\tau), \ldots, y(t-(m-1)\tau))$, then necessarily the next point on the reconstruction space trajectory is the same for both times (and the trajectories in both spaces are closed curves, as in Figure 5). That is, $y(t^\star+\tau) = y(t+\tau)$. If $t^\star$ happens to be the current time, then a match of any previous $m$ consecutive time series values (with the $m$-tuple ending with $y(t^\star)$) provides a means of predicting $y(t^\star+\tau)$: simply read off the known next value of the time series, $y(t+\tau)$.

### *2.4 Conclusion*

This chapter has introduced the concepts of chaotic dynamical systems and fractal properties. It is stressed that for all but a few simple examples, numerical techniques must be relied upon to reveal the presence or absence of chaotic dynamics, and the results of these

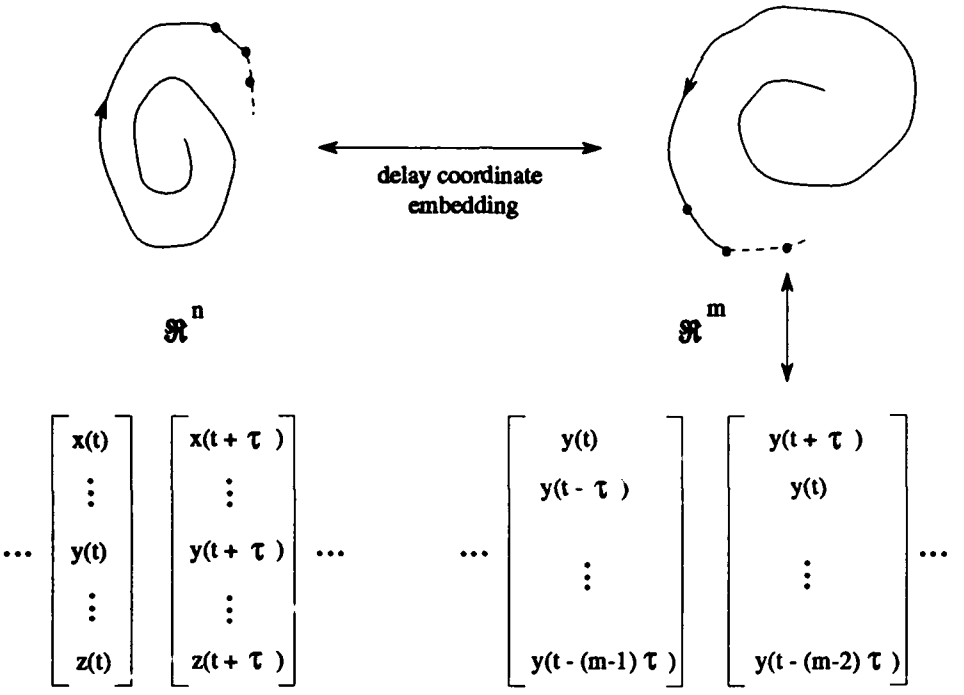



Figure 9. An Embedding in Reconstruction Space $\Re^m$

techniques are often open to subjective interpretation. The terminology *chaotic time series* is herein understood to designate a time series which exhibits a unique finite fractal dimension. In this work, this will generally mean that Grassberger and Procaccia analyses reveal nearly linear log-log plots.

The following chapter will build on the important embedding theorem to develop two modifications of the local linear prediction method introduced by Farmer.

# *III. Further Localizing Local Linear Prediction*

## *3.1 Introduction*

This chapter develops some modifications of the local linear prediction method introduced by Farmer in 1987 (8). Section 3.2 reviews the basic method. Section 3.3 provides insight into the method by demonstrating some conditions under which linear prediction is exact. Section 3.4 reviews some important enhancements to the local linear prediction method which were introduced by Casdagli (4). He showed how to determine appropriate data-dependent parameters for use with the method. These parameters consist of the best pair $(m, k)$ of embedding dimension $m$ and number $k$ of nearest neighbors to use *on average* for prediction from anywhere within a reserved portion (called the "testing set") of the available time series data. It may seem that prediction accuracy could be enhanced by using the value of $k$ found optimal for a single point in the embedding space which is nearest the desired prediction point. It is shown in Section 3.5, however, that such a localization of parameter estimation often resulted in reduced prediction accuracy. The next two sections describe the two pruning algorithms developed during this research, and provide support for the contention that they often enhance prediction accuracy over pure local linear prediction for time series which are not extremely stochastic. Section 3.6 explains a method for predicting from a given point by pruning from a set of its neighbors (of globally determined optimal size, or slightly larger) those neighbors which are least typical. Section 3.7 then gives a method of prediction from a given point based on two optimal time delays associated with the given data. Sets of nearest neighbors are found corresponding to each of the delays, and one of the sets is pruned based on the overlap of its constituent time intervals with those of the other set. Three or more delays could be used with a suitable overlap criterion. Some time series not described previously in the chapter, including some financial time series, are considered in Section 3.8.

### 3.2 *Local Linear Prediction*

As noted earlier, any time series obtained from a dynamical system having a fractal dimension may be delay coordinate embedded into a Euclidean space of sufficiently high dimension that the embedding is dynamically equivalent to the original system. Suppose the time series $x(t), t = 1, 2, \ldots$ corresponds to any component of such a system, and that the system has fractal dimension $d$. The embedding theorem implies that if $m$ is an integer greater than $2d$, then each set of points $\{x(t-1), x(t-2), \ldots, x(t-m)\}$ uniquely determines a next point $x(t)$.

The embedding theorem gives a sufficient condition for dynamic equivalence, not a necessary one. That is, there may be a space of integer dimension less than $2d$ which also accepts an embedding of the original dynamics. This is, in fact, the case for a time series extracted from the Lorenz equations, since the Lorenz attractor has a fractal dimension of about 2.05. Consider a time series obtained by sampling the function depicted in Figure 10. Suppose it is possible to embed the dynamical system from which the time series was obtained

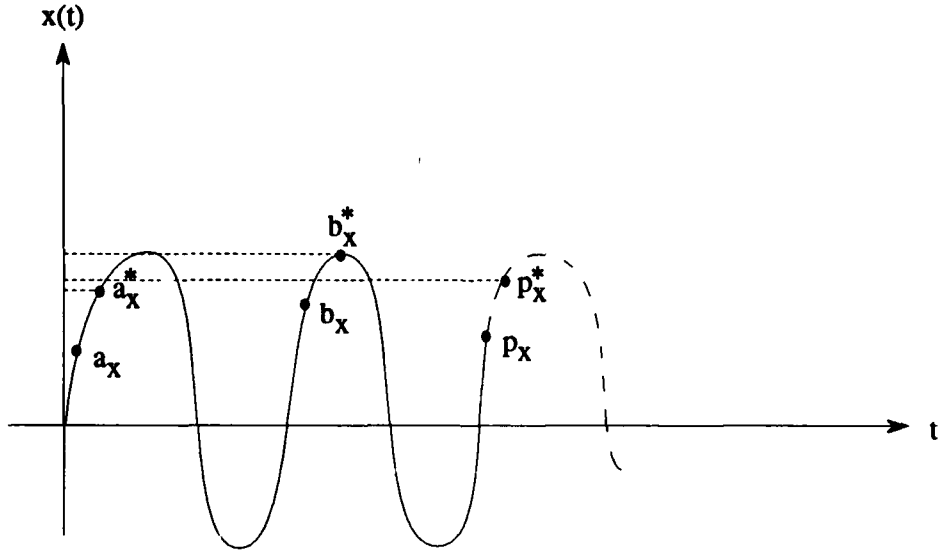



Figure 10. Selected (Value, Next Value) Pairs of a Time Series

in one dimension, so that its reconstruction space phase portrait is a function of one variable

as depicted in Figure 11. In Figure 11, every present value $y$ has a unique next value $y^*$,

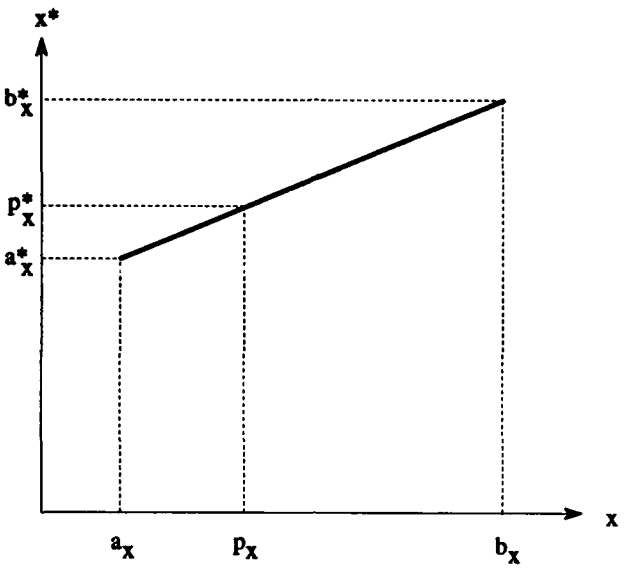



Figure 11. Selected Next Values Over a One-Dimensional Embedding Space

although only three such (value, next value) pairs are shown.

Suppose the last known value of the time series is $p_x$, and it is desired to predict its value $p_x^*$ at the end of the next sample period. One method of prediction is to perform a linear regression using the nearest neighbors to the point $p_x$, i.e., using the time series values closest to $p_x$ (regardless of their time indices) - in this example, the points $a_x$ and $b_x$. That is, a line is formed joining the pairs of points $(a_x, a_x^*)$ and $(b_x, b_x^*)$ in $\Re^2$. The value $p_x^*$ assumed by this regression line at $p_x$ is the predicted value.

Linear regression generalizes easily to Euclidean spaces of arbitrary positive integer dimension (4:307). It typically involves using more than the minimum number of nearest neighbors necessary to establish a hyperplane; in the case of Figure 11, more than the two points nearest $p_x$. In these cases, a hyperplane is fit to the available (neighbor, next value) pairs in such a way as to minimize the resulting squared error. In essence, a least-squares-optimal hyperplane is used to approximate the (neighbor, next value) surface in the neighborhood of the last known portion of the time series. See Figure 12, for example, in which the next value

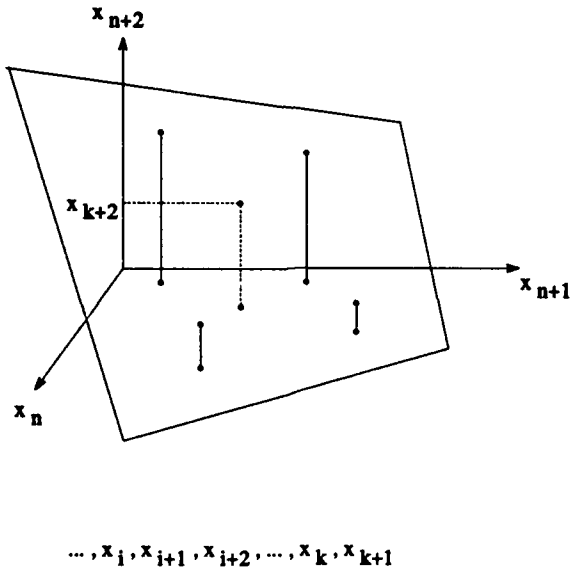



Figure 12. Local Linear Prediction with a Two-Dimensional Embedding Space

$x_{k+2}$ of a time series is predicted using a plane fit through four (neighbor, next value) points corresponding to the four neighbors closest to the point $(x_k, x_{k+1})$.

### *3.3 Some Sufficient Conditions for Perfect Linear Predictability*

Consider again the prediction of the first unknown value $p_x^*$ of the time series depicted in Figure 10. Under what circumstances will linear regression prediction yield the exact value?

A perfect linear regression prediction is shown in Figure 11. That is, if the value of $p_x^*$ is as indicated, then linear regression gives its value exactly. The only requirement for the points $(b_x, b_x^*)$, $(p_x, p_x^*)$, and $(a_x, a_x^*)$ to be collinear is that the slopes of the lines joining $(b_x, b_x^*)$ to $(p_x, p_x^*)$ and $(p_x, p_x^*)$ to $(a_x, a_x^*)$ be equal. That is,

$$\frac{b_x^* - p_x^*}{b_x - p_x} = \frac{p_x^* - a_x^*}{p_x - a_x} \tag{3}$$

The time series from the sampled function $x(t)$ is assumed chaotic. This means that the function $x(t)$ represents a component of a chaotic dynamical system. For simplicity, assume

this system has an attractor which can be represented in two dimensions. Assumptions about the attractor which make Equation 3 valid will now be considered.

Suppose the attractor is a dense cluster of loops that are nearly concentric circles. This is simplified further in Figure 13 by showing three collinear points $a, p$ and $b$ on three concentric

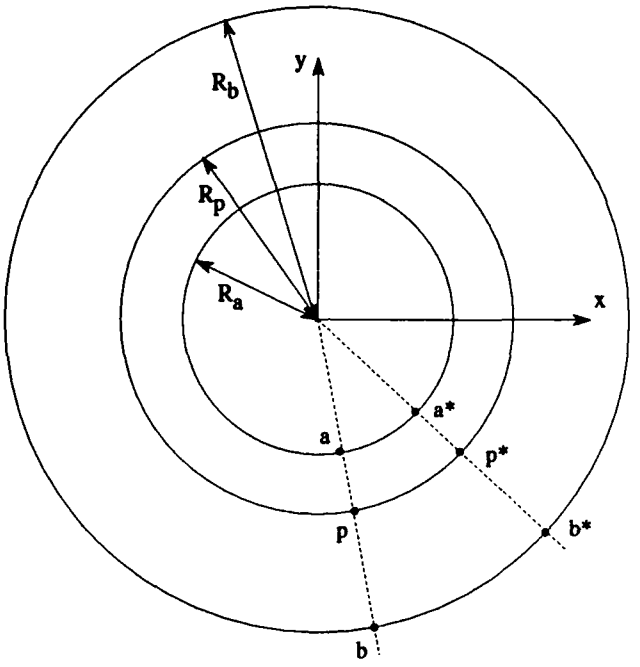



Figure 13. An Attractor as Concentric Racetracks

circular loops (shown disconnected) of the attractor. Suppose that their respective succeeding points $a^*, p^*$ and $b^*$ evolved counterclockwise and in such a way as to remain in perfect alignment. If the lines $ab$ and $a^*b^*$ intersect at the center $(x, y) = (0, 0)$ of the imaginary attractor, then the ratios of their $x$-component increments remain unchanged. That is,

$$\frac{b_x - p_x}{p_x - a_x} = \frac{b_x^* - p_x^*}{p_x^* - a_x^*} \tag{4}$$

since each ratio is equal to

$$\frac{R_b - R_p}{R_p - R_a}$$

by similar triangles. Since Equation 4 is simply a cross multiplication of Equation 3, this assumed configuration of points results in *perfect* linear regression predictability. This configuration is analogous to being near a "center" of an attractor. Relatively slight perturbations of this assumed geometry can seriously degrade the equality of the ratios of $x$-component increments. For example, consider the effect of a shift of the intersection of the lines $ab$ and $a^*b^*$ to the point $-R_a/4$ on the $y$-axis.

On the other hand, if the points under consideration are relatively far from any "center" of an attractor, an adequate model of the local dynamics might be as shown in Figure 14. The line segments $aa^*$, $pp^*$ and $bb^*$ are assumed parallel, as are the lines through the sets of

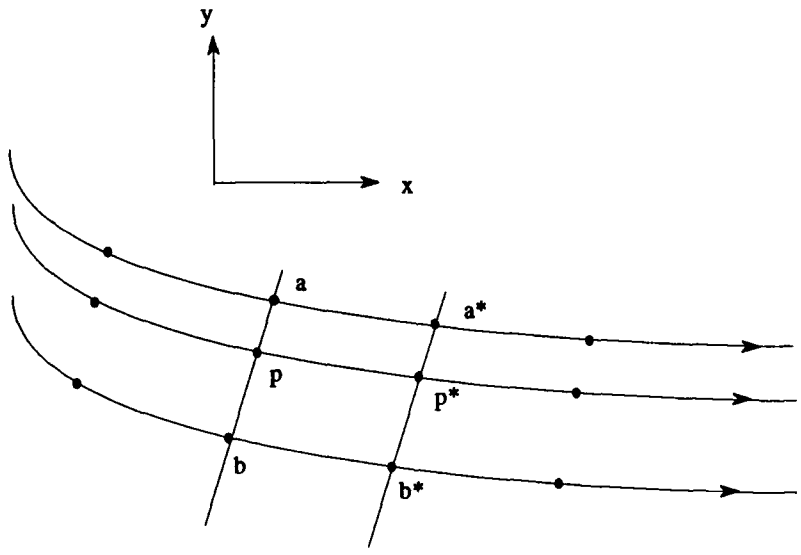



Figure 14. Parallel Segments of an Attractor

collinear points $a, p$ and $b$, and $a^*, p^*$ and $b^*$. In this case, too, the ratios of $x$-component increments remain unchanged, so linear regression predictability is again *perfect*.

In general, linear regression involves using more than the minimum number of points necessary to establish a hyperplane. An $n$-dimensional hyperplane requires at least $n + 1$ points for its linear regression determination (29:34), but often more than $n + 1$ points are available. In the present case, linear regression often involves using more than two points

to establish a linear regression line. If an additional neighbor $c$ of the point $p$ is used, and if the point $(c_x, c_x^*)$ is collinear with the points $(a_x, a_x^*)$ and $(b_x, b_x^*)$ of Figure 11, then the rule for perfect linear regression predictability, Equation 3, is unchanged. That is, all $b$'s, say, in Equation 3 can simply be replaced with $c$'s. However, if $(c_x, c_x^*)$ is not collinear with the points $(a_x, a_x^*)$ and $(b_x, b_x^*)$, then perfect linear predictability equates to the perfect fit of the point $(p_x, p_x^*)$ to the regression line determined by the three points $(a_x, a_x^*)$, $(b_x, b_x^*)$ and $(c_x, c_x^*)$. Assumptions of perfect linear predictability always reduce to assumptions on the unknown value $p_x^*$.

This development generalizes to spaces of integer dimension greater than two. The intuition gained is that if past local trajectory segments closely resemble the present segment, then local linear prediction is likely to be quite accurate.

### *3.4 Identification of Optimal Parameters*

Given a time series for which local linear prediction is to be implemented, a number of parameters must first be determined. It is assumed that the time series is fractal, that is, that a correlation dimension $d$ has been identified for it. The embedding theorem guarantees that the time series may be embedded in a space of any integer dimension $m$ greater than $2d$. There may be, however, a space of dimension less than or equal to $2d$ in which the time series can also be embedded. In any case, for purposes of linear prediction, some advantage may be realized by choosing one sufficiently large embedding dimension over another.

Furthermore, whatever embedding dimension $m$ is chosen, some number $k$ of neighbors nearest the prediction point in the embedding space (i.e., the final $m$-tuple of the time series) must be chosen. In general, a best value of $k$ will depend on the chosen value of $m$.

Assuming a limitless supply of time series data, what is the best number $N$ of time series values to use? Furthermore, should *every* time series value be used, or should gaps be allowed between them? For example, should only every $\tau^{th}$ time series value be used, where the *delay time* $\tau$ might exceed one? What is the desired prediction horizon - one time unit

into the future, or $T$ time units? These are some of the decisions which need to be made; in practice, they are often made by trial and error.

Martin Casdagli presented a technique for finding both the best $m$ and $k$ for a given time series (4). It is essentially an exhaustive search of all possible $m$'s and $k$'s for the pair of values $(m, k)$ which yields the lowest prediction error over a restricted portion of the time series.

Although Casdagli does not elaborate the point, some attempt should be made to "detrend" the given data, ideally to create a time series having locally identical statistics. For example, many financial time series exhibit strong upward trends, and the final embedding point of the raw data from such a time series may have too few neighbors to perform a meaningful regression. Perhaps the simplest way to detrend the data is to replace the given data values $x_i$ with their differences from the succeeding values; that is, the given time series $x_1, x_2, x_3, \ldots$ is replaced with the time series $y_1 = x_2 - x_1, y_2 = x_3 - x_2, \ldots$ (24:257). The resulting time series is then normalized to zero mean and unit variance (5:351). That is, each time series value $y_i$ is then replaced with $(y_i - \mu)/\sigma$ where $\mu$ is the average value of all the $y_j$ and $\sigma$ is their standard deviation. This was the approach taken throughout most of this research. Hereafter, the resulting normalized detrended time series is denoted $x_1, x_2, x_3, \ldots, x_N$. The following steps were then applied, preserving Casdagli's notation. Figure 15 illustrates most of the quantities involved.

(a) Two distinct portions of the time series are identified, one consisting of the first $N_f$ time series values $x_1, x_2, \ldots, x_{N_f}$ (called the fitting set) and the other of the final $N_t$ values $x_{N_f+1}, x_{N_f+2}, \ldots, x_{N_f+N_t}$ (called the testing set). Thus $N_f + N_t = N$.

(b) Some small value of $m$ is selected (typically three or four; it will be incremented), some delay time $\tau$, and some prediction horizon $T$. In general, $\tau$ and $T$ need not be equal, but they were always taken equal in this work.

(c) Now the $m$-tuple $\mathbf{x}_i$ ending with $x_i$, $i \geq N_f$, is chosen for a $T$-step ahead forecasting test. The index $i$ will be incremented by one across as much of the testing set as possible; that

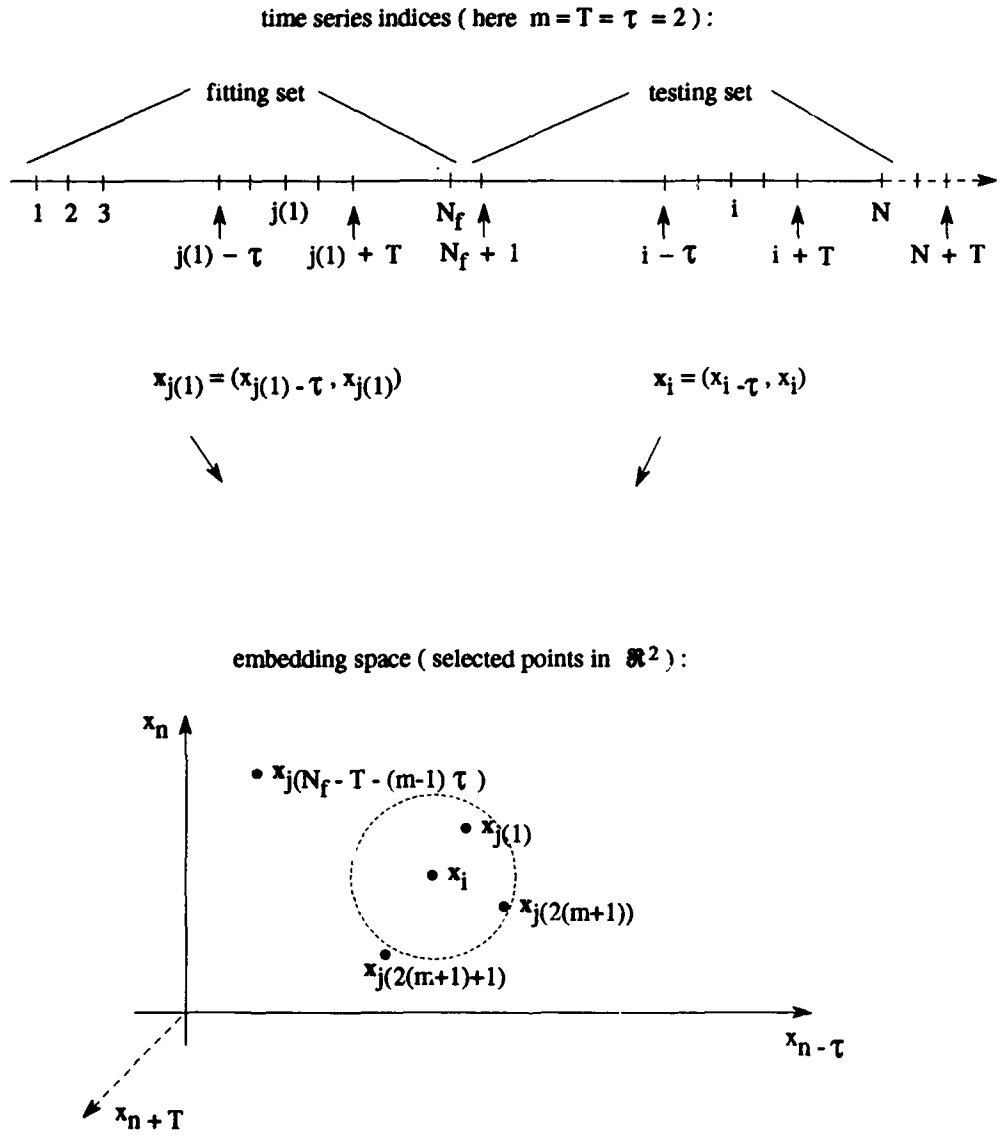



Figure 15. Illustration of Embedding Parameters

is, until $i+T > N$. The $m$-tuples $\mathbf{x}_q$ are called delay vectors, or sometimes *fitting* vectors (if $x_q$ is in the fitting set) or *test* vectors (if $x_q$ is in the testing set).

(d) The distances $d_{ij}$ from the test vector $\mathbf{x}_i$ to the fitting vectors $\mathbf{x}_j$ are computed, using the maximum norm (i.e., maximum of the absolute differences of paired component values) for computational efficiency. Here $j$ ranges over the set of integers in the interval $[1+(m-1)\tau, N_f - T]$.

(e) The distances $d_{ij}$ are ordered so that the relative proximities of the fitting vector neighbors to $\mathbf{x}_i$ can be determined. The neighbor nearest $\mathbf{x}_i$ is labeled $\mathbf{x}_{j(1)}$; the next nearest neighbor is labeled $\mathbf{x}_{j(2)}$; etc. A least squares hyperplane in $\Re^{m+1}$ is then fitted to the $m+1$-tuples formed from the $k$ nearest neighbors in $\Re^m$ and their respective "forecasted" values. That is, the parameters $\alpha_s$ are determined by least squares from the following affine model:

$$x_{j(l)+T} \approx \alpha_0 + \sum_{n=1}^{m} \alpha_n x_{j(l)-(n-1)\tau}, \qquad l = 1, \ldots, k \tag{5}$$

The number $k$ of nearest neighbors is incremented from $2(m+1)$ to $N_f - T - (m-1)\tau$, and a prediction error is calculated for each $k$ (see step (f)). Exponential spacing of $k$'s is usually adequate; for example, the values for $k$ may be taken as the set

$$\{2(m+1), 2(m+1)+2^0, 2(m+1)+2^1, 2(m+1)+2^2, \ldots, \leq N_f - T - (m-1)\tau\}$$

Each of the $k$ approximations in Equation 5 represents an error which depends on the parameters $\alpha_0, \alpha_1, \ldots, \alpha_m$. Squaring these errors (i.e., the differences of the left- and right-hand sides of these approximations), and summing them, gives an error function dependent on these $m+1$ parameters. Setting the partial derivatives of this error function to zero gives a system of $m+1$ equations (the 'normal" equations) in $m+1$ unknowns. The solution of this system (assuming it has a unique solution) is the set of values $\alpha_s$ which give the desired least squares hyperplane. The normal equations may be updated recursively as $k$ is increased. Let $\tilde{m} = m-1$. In vector notation, the system of equations actually being solved is $\boldsymbol{A\alpha} = \boldsymbol{b}$

where **A** is the matrix

$$\begin{bmatrix} k & \sum_{l=1}^{k} x_{j(l)} & \sum_{l=1}^{k} x_{j(l)-\tau} & \cdots & \sum_{l=1}^{k} x_{j(l)-\tilde{m}\tau} \\ \sum_{l=1}^{k} x_{j(l)} & \sum_{l=1}^{k} x_{j(l)}x_{j(l)} & \sum_{l=1}^{k} x_{j(l)}x_{j(l)-\tau} & \cdots & \sum_{l=1}^{k} x_{j(l)}x_{j(l)-\tilde{m}\tau} \\ \sum_{l=1}^{k} x_{j(l)-\tau} & \sum_{l=1}^{k} x_{j(l)-\tau}x_{j(l)} & \sum_{l=1}^{k} x_{j(l)-\tau}x_{j(l)-\tau} & \cdots & \sum_{l=1}^{k} x_{j(l)-\tau}x_{j(l)-\tilde{m}\tau} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{l=1}^{k} x_{j(l)-\tilde{m}\tau} & \sum_{l=1}^{k} x_{j(l)-\tilde{m}\tau}x_{j(l)} & \sum_{l=1}^{k} x_{j(l)-\tilde{m}\tau}x_{j(l)-\tau} & \cdots & \sum_{l=1}^{k} x_{j(l)-\tilde{m}\tau}^{2} \end{bmatrix}$$

and where the vectors $\boldsymbol{\alpha}$ and $\mathbf{b}$ are given by

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} \sum_{l=1}^{k} x_{j(l)+T} \\ \sum_{l=1}^{k} x_{j(l)}x_{j(l)+T} \\ \sum_{l=1}^{k} x_{j(l)-\tau}x_{j(l)+T} \\ \vdots \\ \sum_{l=1}^{k} x_{j(l)-\tilde{m}\tau}x_{j(l)+T} \end{bmatrix}$$

Notice that all of the summations can be updated as $k$ is increased simply by adding additional terms. The actual solution of the system $A\alpha = b$ is facilitated using LU decomposition; see e.g. (29:44–48) and Appendix C.

(f) Using the hyperplane determined by Equation 5, a $T$-step ahead forecast $\hat{x}_{i+T}(k)$ is made for the test vector $\mathbf{x}_i$ and for all applicable $k$; that is, the estimated value $\hat{x}_{i+T}(k)$ of $x_{i+T}$ is calculated as

$$\hat{x}_{i+T}(k) = \alpha_0 + \sum_{n=1}^{m} \alpha_n x_{i-(n-1)\tau}$$

For $N_f \leq i \leq N - T$, the error $e_i(k) = |\hat{x}_{i+T}(k) - x_{i+T}|$ is computed.

(g) Steps (c)-(f) are repeated for all applicable $i$ and the normalized root-mean-square (RMS) forecasting error $E_m(k)$ is calculated as

$$E_m(k) = \{[\sum_{i=N_f}^{N-T} e_i^2(k)]/(N_t - T + 1)\}^{1/2}/\sigma$$

where $\sigma$ is the standard deviation of the time series ($\sigma = 1$ for time series which have been normalized to unit variance). If the time series' mean $\mu$ and variance $\sigma^2$ are shared by the testing set data, then the trivial predictions $\hat{x}_{i+T}(k) = \mu$ give a normalized RMS forecasting error of one. On the other hand, if the predictions $\hat{x}_{i+T}(k)$ are perfect, then $E_m(k) = 0$.

(h) Finally, the embedding dimension $m$ is varied and plots are made of $E_m(k)$ versus $k$.

The algorithm described above (with minor modifications) has recently been named the DVS (for deterministic versus stochastic) algorithm (5). The name derives from the fact that the shapes of the resulting plots can provide evidence of low dimensional *deterministic* chaos, or of high dimensional or *stochastic* dynamics. Low dimensional chaos is typically characterized by U-shaped or monotonically increasing plots whose minimum $E_m(k)$ values are small and occur at low values of $k$. High dimensional or stochastic behavior is often indicated by relatively large minimum $E_m(k)$ values occurring at high $k$ values. Compare for example Figure 16, which represents data derived from the low dimensional Henon attractor, and Figure 17, which may be surmised to represent data of higher dimension or more stochastic origin. In fact, Figure 17 was obtained from a time series derived from views taken of a military vehicle by an observer following a raster-style traversal of a viewing sphere (see Chapter IV).

The absolute minimum value attained by the $E_m(k)$ curves provide the best $m$ and $k$ values to use on average for local linear prediction within the testing set; see for example Figure 17, where the best $m$ is 10 and the best $k$ is 86.

The normal equations of the DVS algorithm do not necessarily admit a unique solution; basically, three (or more) points in $\Re^3$ do not determine a unique plane if the points happen

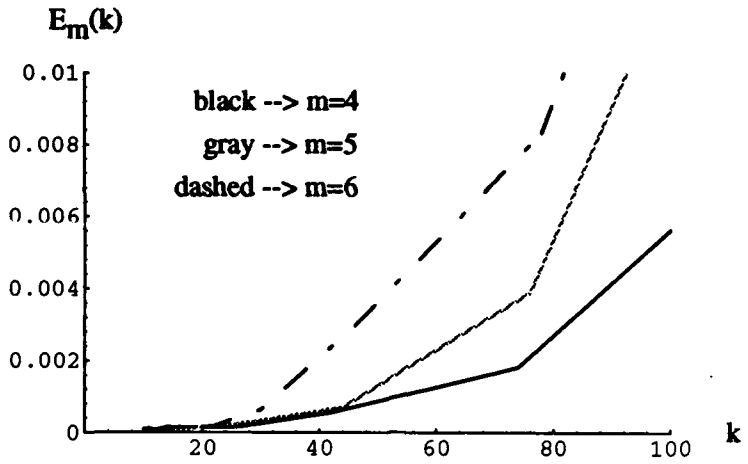



Figure 16. DVS Plots of Low Dimensional Data (from Henon map)

to be collinear (the idea generalizes to higher dimensional spaces). Furthermore, it is possible that some of the points in the various embedding spaces $\Re^m$ may be revisited from different portions of the time series; in the worst case, for example, the $k$ nearest neighbors of a point in $\Re^m$ might all be identical. The latter condition was not often observed in this work; the embedded data was examined for uniqueness before applying the DVS algorithm, and in the few cases where overlap existed for small values of $m$, taking slightly larger values of $m$ eliminated the problem. The former condition is more difficult to screen, but normal equations which were not uniquely solvable (ie, the condition of having a noninvertible matrix **A**) seldom arose in this research. It is felt that taking $k$ greater than or equal to $2(m+1)$ in step (e), rather than merely greater than or equal to the absolute minimum $m + 1$, served to help assure unique solvability, by decreasing the likelihood of inadvertently creating noninvertible matrices **A**.

### *3.5 Using Locally Best Number of Nearest Neighbors*

Suppose that one has used the DVS algorithm with a particular time series and found the optimal $m$ and $k$ values for a particular choice of fitting and testing set sizes. Suppose furthermore that a prediction is desired of the time series value at time $N_f + N_t + T$. One

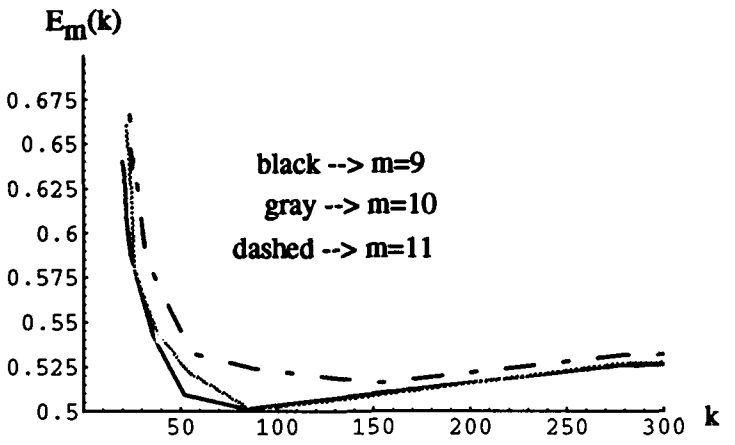



Figure 17. DVS Plots of Higher Dimensional Data (from raster traversal)

clear choice of prediction methodology is to apply local linear prediction at the time $N_f + N_t$, using the optimal $m$ and $k$; that is, simply extend the testing set $m$-tuples to the the most recent one and allow the algorithm to compute one more prediction. Since this requires very little modification of the DVS algorithm, it will be referred to as DVS prediction.

Recall, however, that the optimal $(m, k)$ pair was found *globally* over the entire testing set. Suppose that instead of using the number $k$ of nearest neighbors found optimal for the entire testing set, one used instead the number of nearest neighbors found optimal for predicting from the one $m$-tuple in the testing set which is closest to the $m$-tuple ending at $N_f + N_t$. In terms of Figure 15, the prediction from the $m$-tuple $\mathbf{x}_i$, where $i = N_f + N_t$, is based on the number of neighbors found best for predicting from the $m$-tuple $\mathbf{x}_{j(1)}$ where now $x_{j(1)}$ is understood to be in the testing set rather than the fitting set. The intuition is that different regions of the attractor may be populated more densely with embedded points than are other regions, and more populated regions may predict more accurately with fewer numbers of nearest neighbors determining the linear regression hyperplane. That is, the effect of including neighbors which are too far away may be to skew the prediction hyperplane from its optimal orientation. Using only the number of nearest neighbors found *locally* optimal may reduce

the skewing as compared to using the number of neighbors found *globally* optimal. Similarly, sparsely populated regions may predict more accurately using more nearest neighbors.

Mixed results were obtained when this technique was used. It was applied to the time series "A.dat" used in the 1991 Santa Fe Institute's time series prediction competition. This data was obtained from an infrared laser experiment. It is arguably low-dimensional chaotic and noise-corrupted (41:32,325). Figure 18 shows the 1000 data values made available to the

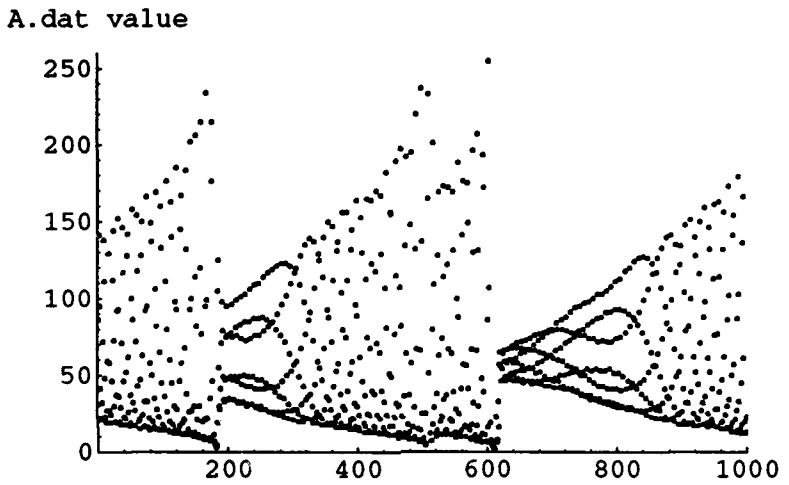



Figure 18. Experimentally Obtained Laser Data

contestants. This data was used in this research without differencing or normalization. The DVS algorithm was applied to this data using $N_f = 800$ and ten successive values of $N_t$ beginning with $N_t = 180$. In it, the parameters used were $m = 4$ and $T = \tau = 1$ (although a smoothing advantage may have been realized by taking $T = \tau > 1$); the globally best number $k$ of nearest neighbors was recomputed with each new value of $N_t$. The predictions are summarized in Table 1. Notice that although the algorithm which used the locally best number of nearest neighbors had a far higher total error, it actually came closer to the correct result than the DVS algorithm half the time.

A similar comparison was made of the two algorithms using differenced and normalized data from the Lorenz equations. Once again, the DVS algorithm had a lower total error, but

Table 1. Laser Data Predictions

| time series index | true val | DVS preds | locally best k preds |
|---|---|---|---|
| 981 | 57.0000 | 57.0509 | 58.5805 |
| 982 | 21.0000 | 20.0848 | 20.3972 |
| 983 | 13.0000 | 13.3638 | 13.5735 |
| 984 | 14.0000 | 13.5747 | 13.6128 |
| 985 | 27.0000 | 26.9606 | 37.6951 |
| 986 | 87.0000 | 83.8718 | 83.7341 |
| 987 | 179.0000 | 182.4079 | 178.1631 |
| 988 | 103.0000 | 104.2120 | 102.7556 |
| 989 | 33.0000 | 31.3810 | 31.6529 |
| 990 | 15.0000 | 15.1364 | 15.1646 |
| RMS error using DVS preds = 1.6329 | | | |
| RMS error using locally best k preds = 3.6193 | | | |

this time only slightly lower. The DVS algorithm was more clearly superior in a test of data generated from the Henon map.

Prediction using the optimal $k$ for the one embedded testing set vector nearest the prediction vector resulted in increased RMS prediction error compared to the DVS algorithm. This approach was therefore abandoned. It may be possible to improve upon the approach by using an averaged value of the best $k$'s for several nearest embedded testing set vectors, rather than just one. This modification may reduce the likelihood of outlier predictions such as occurred at time series index 985 in Table 1.

### *3.6 Pruning by Variance from Regression Hyperplane*

The DVS algorithm often supplies a larger number of nearest neighbors than necessary for a unique solution of the normal equations. It was found that a sort of local filtering can often yield increased prediction accuracy by eliminating from the utilized set of nearest neighbors some which might be considered noise corrupted.

Recall that local linear prediction fits a least squares hyperplane to the (neighbor, next value) surface in the vicinity of the last known $m$-tuple of the time series. Any (neighbor, next value) pairs which depart radically from the presumed reasonably smooth (neighbor, next

value) surface may skew the computed least squares hyperplane and detract from the accuracy of the prediction. Consider Figure 19, in which it is presumed that the DVS algorithm yielded

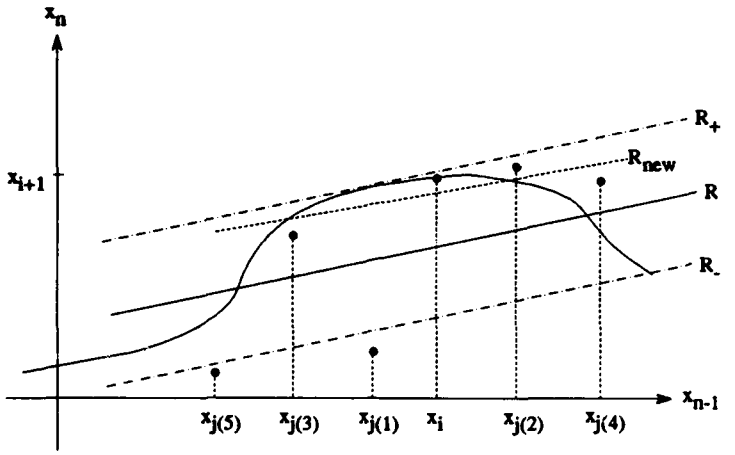



Figure 19. Pruning Nearest Neighbors

an optimum embedding dimension $m = 1$ and number of nearest neighbors $k = 5$. The presumed noiseless (neighbor, next value) surface is shown as a smooth curve. The linear regression hyperplane determined by the five neighbors nearest $x_i$ is actually a line when $m = 1$ and is here denoted $R$. Notice that the "noisy" neighbors $x_{j(1)}$ and $x_{j(5)}$ pull $R$ away from the true next time series value $x_{i+1}$. If these two points are deleted from the set of nearest neighbors and a new linear regression line $R_{\text{new}}$ is determined using only the remaining three neighbors, then the error in the predicted value $\hat{x}_{i+1}$ (found now on $R_{\text{new}}$ instead of on $R$) is reduced. Some criterion must be adopted to determine how far away from $R$ the (neighbor, next value) points must be to warrant exclusion. In this work, the average $\mu$ and standard deviation $\sigma$ of the distances of all next values from $R$ was determined, and points falling outside $\pm(\mu + \gamma\sigma)$ bounds of $R$ were excluded. Here the exclusion parameter $\gamma$ is a real number of small magnitude, typically between zero and four. The exclusion zone in Figure 19 is the region outside the band between $R_+$ and $R_-$.

This idea generalizes to embedding dimensions greater than one. As a prediction algorithm, it will be referred to as pruned outliers prediction. As will be shown, it was

generally found capable of improving prediction accuracy compared to the DVS algorithm. Except for extremely stochastic time series, prediction improvement occurred for most values of $\gamma$ chosen in the range of zero to four. Roughly, pruning anywhere from just one neighbor, up to about ten percent of the DVS-specified neighbors, resulted in improved prediction accuracy. Monitoring the number of neighbors pruned during algorithm implementation allowed verification of compliance with this range. The only data set for which prediction improvement was not observed was financial data consisting of tick-by-tick Swiss franc exchange rates. This data set is examined in Section 3.8.

### *3.7 Pruning by Shared Local History*

Choosing a good delay time $\tau$ for predicting a chaotic time series is not a trivial problem. Because chaos displays sensitive dependence on initial conditions (Appendix A), the delay time should not be too large. On the other hand, if the data is from a finely sampled continuous time system, sampling too closely will result in overlapping points in most reasonably sized embedding spaces. For example, if a sequence contains 20 consecutive identical values because of oversampling, then an embedding space of dimension 19 has at least two repeated points (assuming $\tau = 1$). Furthermore, if a time series is quite noise-corrrupted, taking $\tau$ too small accentuates the effect of the noise. Fraser and Swinney (13) suggest determining a good delay time from the mutual information curve constructed from the given data, although their approach was not taken in this research.

With prediction accuracy the ultimate criterion for the choice of $\tau$, it seems reasonable to choose this parameter directly from multiple runs of the DVS algorithm. For example, once an optimal embedding dimension $m$ and number $k$ of nearest neighbors has been obtained using $\tau = T = 1$, the algorithm can be applied several more times with varying values of $\tau = T$ and with the optimal $m$ fixed. Exhaustive search of the resulting data can reveal which value of $\tau$ gives best results for the selected $m$. Values of embedding dimension near the winning value $m$ could also be so examined, but this approach was not applied in this work. For the data considered in this work, $\tau = 1$ almost always yielded greatest accuracy. This is probably

because the data was not usually given in oversampled form, and any $\tau \geq 2$ necessarily involves discarding some available information. Consider, for example, the data represented by the plot in Figure 20. This plot depicts variations in the blood oxygen concentration of a

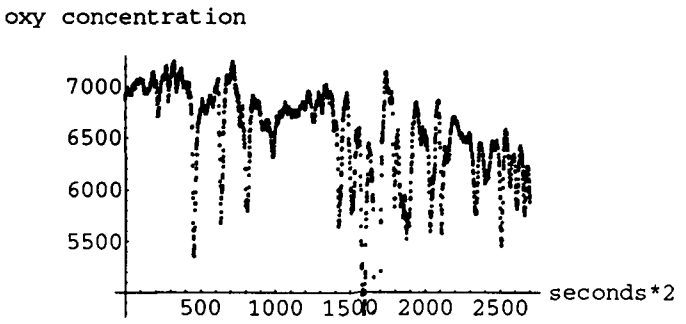



Figure 20. Sleep Apnea Blood Oxygen Concentration

sleep apnea patient. It was taken from data set B1.dat of the Santa Fe Institute's time series prediction competition (41:4). Lines 6781 through 9480 of file B1.dat (2700 data points) were chosen because these values correspond to a lengthy period of uninterrupted stage 2 sleep. After differencing and normalizing this data, DVS plots were made to determine an optimal embedding dimension $m$ and number $k$ of nearest neighbors. Figure 21 shows a few of the

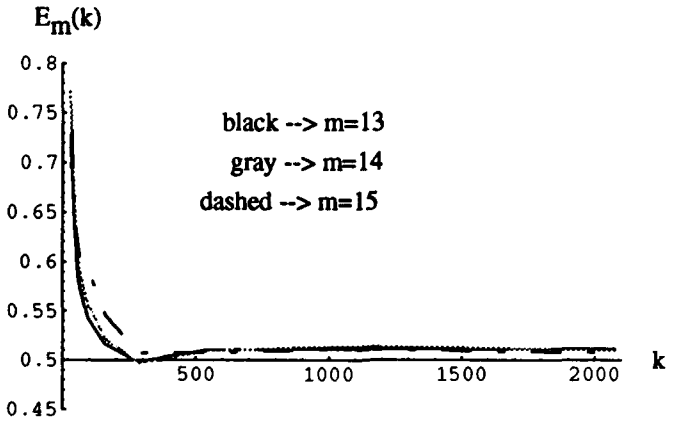



Figure 21. Sleep Apnea DVS Plots

embedding dimensions considered, including the dimension $m = 14$ which yielded the lowest normalized RMS error (at $k = 286$). A commitment is made to the dimension $m = 14$ and a search for values of $\tau$ which might yield lower error is performed by examining the outputs of the DVS algorithm using a range of $\tau$ values (in this case, $\tau = T = 1$ through $\tau = T = 12$). Table 2 shows a subset of this range ($\tau = 4$ through $\tau = 8$); no value of $\tau$ (whether displayed

Table 2. Selected Error Values (Sleep Apnea Data, $m = 14$)

| $k$ | $\tau = 4$ | $\tau = 5$ | $\tau = 6$ | $\tau = 7$ | $\tau = 8$ |
|---|---|---|---|---|---|
| 30 | 0.7704 | 1.0498 | 0.8999 | 0.9554 | 1.0440 |
| 31 | 0.7553 | 1.0100 | 0.8867 | 0.9621 | 1.0181 |
| 32 | 0.7489 | 0.9952 | 0.8839 | 0.9543 | 1.0090 |
| 34 | 0.7514 | 0.9220 | 0.8594 | 0.9161 | 0.9836 |
| 38 | 0.7159 | 0.8789 | 0.7898 | 0.8821 | 0.9413 |
| 46 | 0.6706 | 0.8036 | 0.7713 | 0.8007 | 0.8919 |
| 62 | 0.6580 | 0.7717 | 0.7140 | 0.7317 | 0.8293 |
| 94 | 0.6259 | 0.7213 | 0.6856 | 0.6912 | 0.7316 |
| 158 | 0.6093 | 0.6866 | 0.6606 | 0.6570 | 0.6834 |
| 286 | 0.5885 | 0.6622 | 0.6474 | 0.6491 | 0.6595 |
| 542 | 0.6013 | 0.6516 | 0.6450 | 0.6557 | 0.6604 |
| 1054 | 0.5804 | 0.6437 | 0.6334 | 0.6498 | 0.6625 |
| 2078 | 0.5716 | 0.6451 | 0.6354 | 0.6588 | 0.6389 |

in Table 2 or not) yields a lower value of normalized RMS error than the 0.4974 recorded at $\tau = 1$.

Notice in Table 2 that normalized RMS error often does not increase monotonically with $\tau$ for a fixed value of $k$ (eg, for $k = 158$). It was discovered in the course of this work that for some time series, for a fixed $m$ and fixed $k$, there are two or more values of $\tau$ which give particularly low error values. As will be shown, this was often the case with financial time series. A technique was developed to combine the predictive advantages of two $\tau$'s corresponding to the lowest error values, and it was often able to increase prediction accuracy compared to DVS prediction.

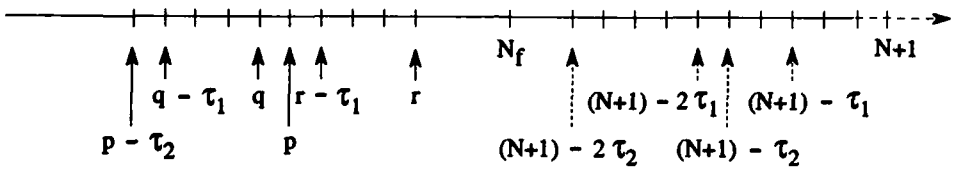
Let $\tau_1$ denote the smaller of the two optimal $\tau$ values and let $\tau_2$ denote the larger. Prediction with this technique proceeds basically the same as DVS prediction, using DVS-

determined values of $m$ and $k$ and with $\tau = \tau_1 = T \triangleq T_1$. Some of the $k$ nearest neighbors are eliminated, however, before the prediction hyperplane is determined. The neighbors pruned correspond to intervals in time which do not overlap intervals relevant to prediction using $\tau_2$. Intuitively, if the data corresponding to a particular interval of time are not prediction-relevant to both short term and long term prediction, then that data might beneficially be eliminated from the set of prediction-determining points in the embedding space. Perhaps the nonoverlapping data represents noise.

Consider Figure 22, for example, in which a prediction of the time series value $x_{N+1}$ at time $N + 1$ is desired. It is assumed in Figure 22 that for the embedding dimension $m = 2$ and some number $k$ of nearest neighbors, $\tau_1 = 3$ and $\tau_2 = 5$ have been found optimal. Prediction occurs from the point $\mathbf{x}_{(N+1)-\tau_1}$ in the $\tau_1$ embedding space using neighbors nearest $\mathbf{x}_{(N+1)-\tau_1}$ in the fitting set. Not all of the $k$ nearest neighbors are used, though. Short solid arrows in Figure 22 indicate the endpoints of intervals in the fitting set corresponding to some of the $k$ points in the $\tau_1$ embedding space which are closest to $\mathbf{x}_{(N+1)-\tau_1}$. The long solid arrows indicate the endpoints of an interval corresponding to one of the $k$ nearest neighbors to the point $(x_{(N+1)-2\tau_2}, x_{(N+1)-\tau_2})$ in its $\tau_2$ embedding space (the $\tau_2$ embedding space is not illustrated). It is assumed that none of the intervals of length five corresponding to the $k$ nearest neighbors of $(x_{(N+1)-2\tau_2}, x_{(N+1)-\tau_2})$ overlap the interval $[r - \tau_1, r]$. Since there is no overlap, the point $\mathbf{x}_r$ is eliminated from the set of points which will determine the prediction hyperplane. On the other hand, the interval $[q - \tau_1, q]$ does overlap the interval $[p - \tau_2, p]$, so the point $\mathbf{x}_q$ is retained. Care is taken that at least $2(m + 1)$ nearest neighbors are retained regardless of overlap, and local linear prediction proceeds as described in Section 3.4.

The algorithm just described will be referred to as overlap prediction. A number of variations of it come readily to mind. There is no inherent need to combine only two optimal $\tau$'s, and various schemes can be devised to eliminate neighbors based on complete or partial isolation of prediction-relevant intervals when intervals of several different lengths are used. Even in the case of only two optimal $\tau$'s, some benefit might be gained by requiring strict inclusion of smaller intervals (as illustrated in Figure 22) rather than merely overlap. Never-

time series indices ( here $m=2$, $\tau_1=T_1=3$ , and $\tau_2=T_2=5$ ) :




$$x_{(N+1)-\tau_1} = (x_{(N+1)-2\tau_1}, x_{(N+1)-\tau_1})$$

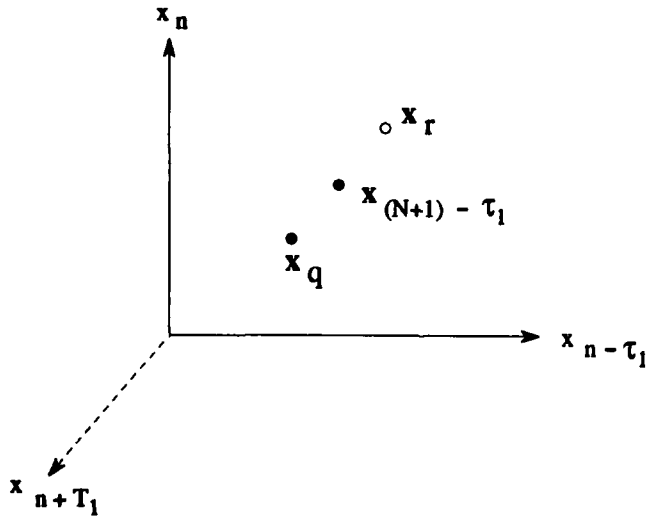



Figure 22. Overlap Prediction

theless, overlap prediction as presented above often gives prediction accuracy improvement for many time series. Table 3 compares DVS predictions, pruned outliers predictions, and

Table 3. Selected Predicted Values (Sleep Apnea Data, $m = 14$)

| ts index | true val | DVS preds | pruned outliers preds | overlap preds |
|---|---|---|---|---|
| 2671 | 1.4624 | 0.4120 | 0.3051 | 0.2894 |
| 2672 | 0.3710 | 1.0757 | 1.1453 | 0.1562 |
| 2673 | 1.3210 | 0.6858 | 0.7350 | 0.3807 |
| 2674 | 0.9370 | 0.7490 | 0.6667 | 0.4318 |
| 2675 | 0.8359 | 0.7923 | 0.7386 | 0.6121 |
| 2676 | 0.3104 | 0.6682 | 0.4319 | -0.0132 |
| 2677 | 0.9167 | 0.8379 | 0.8287 | 0.5865 |
| 2678 | -0.1342 | 0.5722 | 0.4713 | -0.0879 |
| 2679 | 0.0477 | 0.8037 | 0.6235 | 0.6758 |
| 2680 | 0.0679 | 0.2812 | 0.3404 | 0.0634 |
| RMS error | | 0.5737 | 0.5621 | 0.5676 |

overlap predictions for the sleep apnea time series. In applying DVS prediction, the number $k$ of nearest neighbors used was 286, and $\tau$ and $T$ were set to one. In applying pruned outliers prediction, the same values of $k, \tau$ and $T$ were used, and the parameter $\gamma$ which determines the exclusion region was set to two. A smaller number of nearest neighbors was used with overlap prediction; here $k = 158$ was used to avoid swamping the available time with candidate intervals (thereby making pruning unlikely to occur). Although a wider range of $\tau$'s was considered than is shown in Table 2, this table nonetheless illustrates why the values of $\tau_1$ and $\tau_2$ were chosen as four and seven respectively.

### *3.8 Some Other Time Series*

The pruned outliers and overlap prediction algorithms yielded improvement over DVS prediction for many, but not all, types of time series. The laser data discussed in Section 3.5 was differenced and normalized (as described in Section 3.4), then analyzed using the three algorithms. Data generated by the DVS algorithm (with $N_f = 560$ and $N_t = 439$) revealed an optimal embedding dimension of seven and number of nearest neighbors $k = 17$. Further runs of this algorithm with $m$ fixed at seven and with varying delay times $\tau$ produced

data from which a number 48 of nearest neighbors was selected for overlap prediction using $\tau_1 = 4$ and $\tau_2 = 7$. For the pruned outliers algorithm, the exclusion parameter $\gamma$ was taken to be 2.0. Ten data values beyond those available to the Santa Fe contestants were utilized for purposes of comparing predictions (these values being the first ten points in the continuation file A.cont). The results are presented in Table 4. Recall that at least $2(m + 1)$ neighbors are

Table 4. Selected Predicted Values (Santa Fe Laser Data, $m = 7$)

| ts index | true val | DVS preds | pruned outliers preds | overlap preds |
|---|---|---|---|---|
| 1000 | 1.0807 | 1.0734 | 1.0734 | 1.1939 |
| 1001 | 2.3363 | 2.2353 | 2.2437 | 2.0421 |
| 1002 | .2321 | -1.0735 | -1.0735 | -0.8828 |
| 1003 | -1.8929 | -1.8870 | -1.8868 | -1.8261 |
| 1004 | -0.4832 | -0.4743 | -0.4706 | -0.5686 |
| 1005 | -0.0647 | -0.0745 | -0.0747 | -0.0662 |
| 1006 | 0.0675 | 0.0743 | 0.0743 | -0.1847 |
| 1007 | 0.3979 | 0.4200 | 0.4200 | 0.4634 |
| 1008 | 1.7195 | 1.6223 | 1.7646 | 1.7197 |
| 1009 | 1.7195 | 1.7033 | 1.7405 | 1.7267 |
| RMS error | | 0.0677 | 0.0609 | 0.1735 |

always retained for determining the prediction hyperplane; in this case, 16. Thus the pruned outliers predictor can only prune at most one neighbor. In fact, with $\gamma = 2.0$, it only pruned one neighbor in six of the ten predictions; the remaining four predictions used all 17 nearest neighbors and were therefore identical with the DVS predictions. Recognizing the lack of "prunability" allowed by the use of the DVS optimal $k = 17$, a larger number (48) of nearest neighbors was used with overlap prediction. Unfortunately, normalized RMS errors escalate quickly as the number of nearest neighbors increases from 17. From $k = 17$ to $k = 48$, there is an increase of normalized RMS error from 0.21 to 0.34 for $\tau = 1$; similar increases exist for all examined $\tau$'s. As Table 4 reveals, overlap prediction cannot overcome the inherent loss of prediction accuracy associated with using too many neighbors.

The laser data just examined is presumed to have relatively few degrees of freedom associated with its generation. Its DVS plots reveal that it is nearer the D (deterministic) than the S (stochastic) extreme. On the other hand, certain types of financial data have higher

numbers of nearest neighbors associated with their optimal prediction, even relative to the sometimes larger embedding dimensions. Accordingly, their DVS plots reveal more stochastic behavior, or higher dimensional dynamics. Such time series are less accurately predictable than those coming more from the deterministic extreme, but it is easier to enhance whatever predictability there is using the pruned outliers and overlap prediction algorithms.

Figure 23 shows daily opening bids for Standard and Poor's 500 futures contracts for

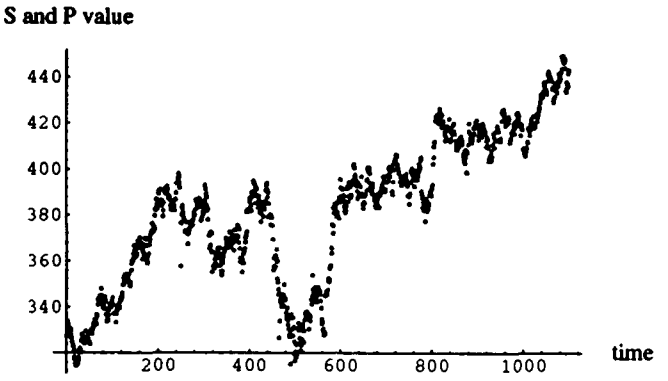



Figure 23. Standard and Poor's Bid Prices

the period from 20 October 1988 to 26 February 1993; it contains a total of 1100 data points. These data were differenced, then normalized to zero mean and unit variance (as described in Section 3.4). The resulting time series (containing 1099 data points) is shown in Figure 24. This time series was then processed by the DVS algorithm (using $N_f = 990$ and $N_t = 90$)

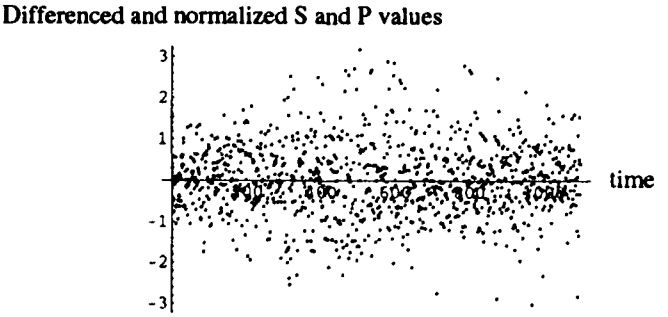



Figure 24. Differenced S and P Data, Normalized to Zero Mean and Unit Variance

to establish optimal values of embedding dimension and number of nearest neighbors. These were determined to be $m = 12$ and $k = 154$ respectively. These values of $m$ and $k$ were also used in pruned outliers prediction, with the exclusion parameter $\gamma = 4.0$ (see Section 3.6). For overlap prediction, the smaller value $m = 3$ was chosen to avoid excessive overlap of intervals; had $m = 12$ been used in overlap prediction with a delay time of $\tau = 11$, for example, then each interval would extend over $12 \times (11 - 1) = 120$ time series values. Since there are only 1099 time series values available, the chances of pruning would be quite small for any moderate number of nearest neighbors. The value $m = 3$ was chosen not only because 3 is small but also because the normalized RMS errors associated with prediction using $m = 3$ (0.69 at $k = 72$, for example) are only slightly higher than those associated with prediction using $m = 12$ (0.67 at $k = 72$). Having chosen to use $m = 3$ with overlap prediction, a good number of nearest neighbors and corresponding good delay times $\tau_1$ and $\tau_2$ were selected. Table 5 shows some of the $\tau$ values considered, and illustrates why the values $k = 72$, $\tau_1 = 3$ and $\tau_2 = 11$ were chosen. Some results of predicting with the various

Table 5. Selected Error Values (Standard and Poor's Data, $m = 3$)

| $k$ | $\tau = 3$ | $\tau = 5$ | $\tau = 7$ | $\tau = 9$ | $\tau = 11$ |
|---|---|---|---|---|---|
| 8 | 0.9829 | 0.8932 | 0.8514 | 1.0856 | 0.9397 |
| 9 | 0.8435 | 0.8607 | 0.8608 | 0.9934 | 0.8805 |
| 10 | 0.8377 | 0.8400 | 0.8035 | 0.8848 | 0.8347 |
| 12 | 0.8498 | 0.8275 | 0.7610 | 0.8634 | 0.8121 |
| 16 | 0.7797 | 0.7688 | 0.7285 | 0.8108 | 0.7290 |
| 24 | 0.7098 | 0.7153 | 0.7258 | 0.7886 | 0.7217 |
| 40 | 0.7005 | 0.7036 | 0.7029 | 0.7545 | 0.6807 |
| 72 | 0.6878 | 0.7007 | 0.6953 | 0.7155 | 0.6432 |
| 136 | 0.6914 | 0.7021 | 0.6889 | 0.7049 | 0.6478 |
| 264 | 0.6905 | 0.6813 | 0.6751 | 0.6951 | 0.6409 |
| 520 | 0.6864 | 0.6865 | 0.6737 | 0.6868 | 0.6475 |

algorithms are presented in Table 6.

Figure 25 shows daily opening prices for the British pound for the period from 19 December 1988 to 27 April 1993. It contains 1101 data points. This data was differenced, then normalized to zero mean and unit variance. The resulting time series (containing 1100

Table 6. Selected Predicted Values (Standard and Poor's Data)

| ts index | true val | DVS preds | pruned outliers preds | overlap preds |
|---|---|---|---|---|
| 1081 | -0.2339 | -0.4028 | -0.3632 | 0.1268 |
| 1082 | 0.7550 | 0.1413 | 0.0693 | 0.1756 |
| 1083 | 0.4932 | 0.2114 | 0.1757 | 0.3174 |
| 1084 | 1.5984 | -0.2228 | -0.0974 | 0.1111 |
| 1085 | -0.1030 | 0.1475 | 0.0605 | -0.0095 |
| 1086 | 0.0424 | 0.1419 | 0.2559 | 0.0398 |
| 1087 | -0.5829 | -0.0624 | -0.0624 | -0.2704 |
| 1088 | -0.3211 | -0.2619 | 0.0126 | -0.0857 |
| 1089 | 0.2896 | 0.1025 | 0.1926 | -0.0146 |
| 1090 | 0.1006 | 0.0146 | 0.0484 | 0.0749 |
| RMS error | | 0.6474 | 0.6269 | 0.5444 |

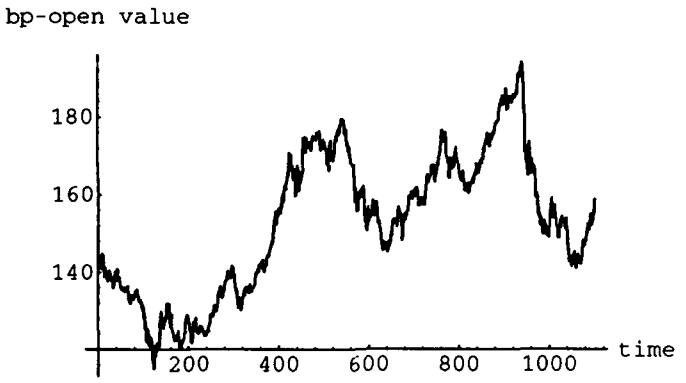



Figure 25. British Pound Opening Bids

data points) was then processed by the DVS algorithm (using $N_f = 990$ and $N_t = 90$) to establish optimal values of embedding dimension and number of nearest neighbors. These were found to be $m = 6$ and $k = 270$ respectively. These values of $m$ and $k$ were also used in pruned outliers prediction, but now with the smaller exclusion parameter $\gamma = 1.0$. The embedding dimension $m = 6$ was also used for overlap prediction; this value of $m$ is considerably smaller than the value $m = 12$ found optimal for the Standard and Poor's data and therefore alleviates excessive overlap caused by very long candidate intervals. Examining the DVS data for a range of $\tau$ values from 1 to 14 resulted in the selection of $k = 46$ as

the baseline number of candidate intervals of lags $\tau_1 = 3$ and $\tau_2 = 14$. Some prediction results using these parameters are presented in Table 7. Although Table 7 reflects the use of

Table 7. Selected Predicted Values (British Pound Data)

| ts index | true val | DVS preds | pruned outliers preds | overlap preds |
|---|---|---|---|---|
| 1081 | -0.1036 | 0.0986 | 0.0962 | 0.2003 |
| 1082 | 1.0478 | 0.1116 | 0.1125 | -0.1885 |
| 1083 | 0.4289 | 0.0594 | -0.0188 | -0.0684 |
| 1084 | 1.5371 | 0.2028 | 0.2133 | -0.1458 |
| 1085 | -1.0246 | -0.0584 | -0.1294 | -0.5622 |
| 1086 | -0.2187 | 0.2834 | 0.3007 | 0.2154 |
| 1087 | 0.1411 | -0.1998 | -0.0121 | -0.0080 |
| 1088 | 0.0835 | 0.0571 | 0.0694 | -0.3890 |
| 1089 | 1.3788 | -0.0694 | -0.0598 | 0.3855 |
| 1090 | 1.2205 | 0.0858 | 0.0930 | 0.2508 |
| RMS error | | 0.8673 | 0.8546 | 0.8529 |

the exclusion parameter $\gamma = 1.0$ in pruned outliers prediction, it was found that several other small positive choices of $\gamma$ also yielded accuracy improvement over DVS prediction.

Figure 26 displays 2000 values of consecutively obtained spatial Fourier magnitude

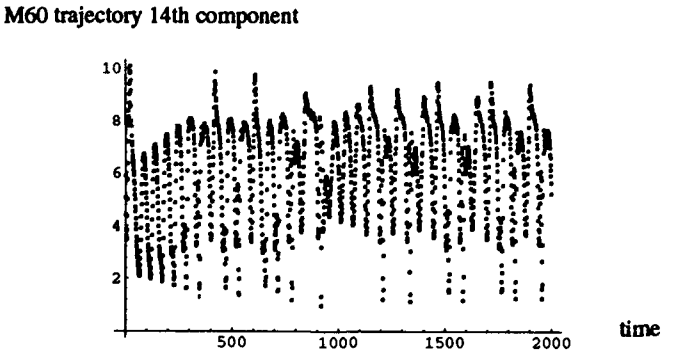



Figure 26. Fourteenth Fourier Component Data from an Apparently Moving M60 Tank

components derived from an M60 tank located at the center of a viewing sphere (see Chapter IV). The tank was viewed while the observer was following a Lorenz-like viewing sphere traversal, and the time series represents fourteenth Fourier components. As usual, these

data were differenced, then normalized to zero mean and unit variance. The resulting time series was then processed by the DVS algorithm (using $N_f = 1820$ and $N_t = 160$) to establish optimal values of embedding dimension and number of nearest neighbors. These were determined to be $m = 9$ and $k = 36$ respectively. These values of $m$ and $k$ were used in both DVS and pruned outliers prediction. The exclusion parameter $\gamma = 2.0$ was selected for pruned outliers prediction. A smaller value of $m$ than 9 was sought for overlap prediction (to avoid excessively long candidate intervals). DVS analyses revealed low prediction errors associated with prediction using $m = 5$, $k = 28$, $\tau_1 = 3$, and $\tau_2 = 5$ (relative to neighboring values of these parameters), so these values were fixed for use in overlap prediction. Some prediction results using these parameters are given in Table 8.

Table 8. Selected Predicted Values (M60 Tank Data)

| ts index | true val | DVS preds | pruned outliers preds | overlap preds |
|---|---|---|---|---|
| 1981 | -0.0860 | -0.0022 | -0.0099 | -0.1011 |
| 1982 | -0.1330 | -0.0197 | -0.0400 | -0.1038 |
| 1983 | 0.0800 | -0.0902 | -0.1256 | -0.0989 |
| 1984 | -0.0964 | -0.0207 | -0.0824 | -0.0453 |
| 1985 | -0.0072 | -0.0272 | 0.0454 | -0.1602 |
| 1986 | -0.2905 | -0.1941 | -0.1920 | -0.1538 |
| 1987 | -0.1957 | -0.1078 | -0.0642 | -0.1997 |
| 1988 | -0.0885 | -0.2379 | -0.1915 | -0.1558 |
| 1989 | -0.3078 | -0.2088 | -0.2951 | -0.3862 |
| 1990 | -0.1371 | -0.3029 | -0.3035 | -0.1956 |
| RMS error | | 0.1147 | 0.1118 | 0.0958 |

It may be that both pruned outliers and overlap prediction (with fortuitous choices of parameters) improve upon DVS prediction by eliminating noise . Not all noisy data is equally amenable to prediction improvement, however. The time series illustrated in Figure 27 is a plot of tick-by-tick quotes of Swiss franc exchange rates, over a period of about two and one half weeks in late 1990 (3000 data points). It represents data set C2.dat of the Santa Fe Institute competition. The values of this data almost always change in multiples of 0.0005, and seven or more contiguous identical values in the data are not uncommon. For this reason, direct embedding of the data results in considerable overlap in all but very high dimensional spaces.

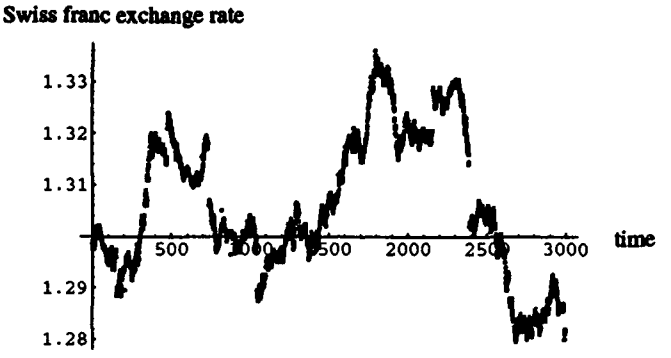



Figure 27. Plot of Santa Fe Institute Swiss Franc Data Set C2.dat

To alleviate this problem, the time series was sampled at every second point, creating a time series with values spaced roughly (but variably) seven minutes apart. The resultant series was differenced, then normalized, giving a time series with 1499 data points. The DVS algorithm was applied using $m$ values from 7 to 14, with $N_f = 1300$ and $N_t = 170$. Analysis of the resulting data indicated best prediction accuracies (such as they were) at quite high values of $k$. The DVS plots for $m = 7$, $m = 8$, and $m = 9$ are shown in Figure 28. Their trailing

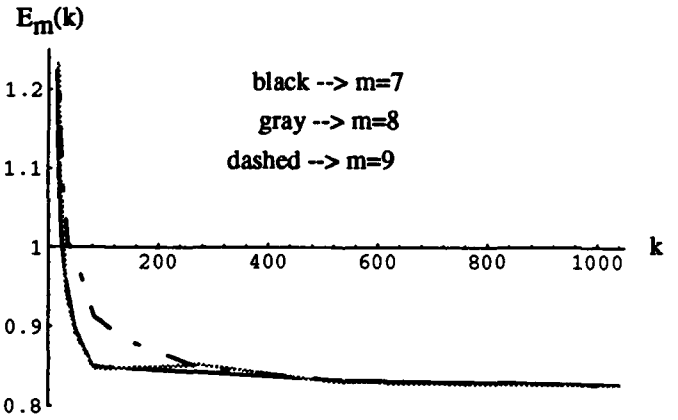



Figure 28. Swiss Franc DVS Plots

tails and high error values (always above 0.82) indicate data near the stochastic (as opposed to deterministic) extreme (4). In other words, these data are likely either very noisy or generated by a dynamical system having a very large number of degrees of freedom. For the range of

$m$'s examined, the least error was found to occur at $m = 8$ with $k = 1042$. These were the values used with DVS prediction. They were also used with pruned outliers prediction, where the exclusion parameter $\gamma = 1.0$ was chosen again. With overlap prediction, $m = 8$ was retained, but $k = 1042$ was so large that many potentially valuable close neighbors would likely be eliminated by the overwhelming number of candidate neighbors, even by neighbors which were quite distant and perhaps error-accentuating. Therefore, when a range of values of $\tau$ were examined (from $\tau = 1$ through $\tau = 16$), the smaller value $k = 82$ was selected, along with $\tau_1 = 4$ and $\tau_2 = 9$. A comparison of the three methodologies is presented in Table 9. Notice that all of the methodologies produce poor predictions (recall that RMS errors

Table 9. Selected Predicted Values (Swiss Franc Data)

| ts index | true val | DVS preds | pruned outliers preds | overlap preds |
|---|---|---|---|---|
| 1471 | -1.3745 | -0.0367 | -0.0049 | -0.1999 |
| 1472 | 0.0047 | -0.0506 | -0.0876 | 0.4973 |
| 1473 | -0.9148 | -0.0369 | -0.0480 | -0.0314 |
| 1474 | -0.4550 | -0.0665 | -0.0690 | -0.0301 |
| 1475 | 0.9242 | -0.0487 | 0.0352 | 0.2161 |
| 1476 | -1.3745 | -0.0961 | -0.0148 | -0.2729 |
| 1477 | 0.0047 | -0.0859 | -0.1370 | 0.1076 |
| 1478 | -0.9148 | -0.0612 | -0.0889 | 0.6342 |
| 1479 | -1.3745 | -0.1714 | 0.0346 | -0.0026 |
| 1480 | 0.9242 | -0.0476 | -0.0034 | 0.1306 |
| RMS error | | 0.9178 | 0.9472 | 0.9602 |

of unity result from merely predicting the average data value). Several different choices of parameters $\gamma$ were tried with pruned outliers prediction, as were several different choices of parameters $k$, $\tau_1$, and $\tau_2$ with overlap prediction. None of these choices improved prediction accuracy over DVS prediction. This is not to say that no choices *could* improve accuracy. Rather, the relative difficulty of improving on DVS prediction accuracy with the Swiss franc data suggests it possesses a very high degree of randomness and may not be a suitable subject for local linear prediction. Mozer was able to attain a slightly lower RMS error (0.859) for 15-minute-ahead predictions using a neural network (26:260).

### *3.9 Conclusion*

The DVS algorithm has been presented in some detail, and its modification to a DVS prediction algorithm has been described. Two extensions of this algorithm have been developed, and some results of their implementations have been presented.

The first algorithm, pruned outliers prediction, excludes from the DVS-determined optimal number of neighbors nearest the prediction point, those which lie farthest from the linear prediction hyperplane. A new hyperplane is calculated based on the remaining neighbors, and the value assumed on this hyperplane at the prediction point becomes the predicted value. This algorithm was able to predict more accurately than DVS prediction for a number of time series examined. For example, prediction improvement was noted using Standard and Poor's financial data, where an RMS error value of 0.627 was found with pruned outliers prediction, as compared to 0.647 with DVS prediction.

The second algorithm, overlap prediction, endeavors to combine the benefits of using one short and one longer time delay. Prediction is based on the shorter of the two time delays, but intervals of time are allowed to be represented in the utilized set of nearest neighbors only if they play a role in both short term and long term prediction. This algorithm was also able to predict more accurately than DVS prediction for some time series examined, particularly financial time series. Prediction of the Standard and Poor's data gave an RMS error of 0.544, better than either of the other algorithms

Both algorithms were therefore shown capable of improving prediction accuracy over DVS prediction for some time series. Other time series seemed less susceptible to improvement. Numerous experiments with Swiss franc exchange rate data yielded decreased prediction accuracy compared to DVS prediction. In one typical experiment, the RMS errors were 0.918, 0.947, and 0.960 for DVS, pruned outliers, and overlap predictions, respectively. It is suspected that the decreased susceptibility to improvement with Swiss franc data may indicate that this data is more stochastic (as implied by Figure 28) than most of the other time series examined.

A total of 60 predictions have been presented (ten for each of six time series) comparing the DVS, pruned outliers, and overlap prediction methods. The total RMS errors associated with all of these 60 predictions are 0.627 for DVS prediction; 0.626 for pruned outliers prediction; and 0.620 for overlap prediction. When the predictions associated with the highly stochastic Swiss franc data are excluded, the 50 remaining predictions reveal total RMS errors of 0.551 for DVS prediction; 0.539 for pruned outliers prediction; and 0.526 for overlap prediction. These numbers suggest that the two algorithms newly developed in this research may offer prediction advantage for time series which are not highly stochastic.

A drawback to both pruned outliers and overlap predictions is that they require selection of parameter(s) which may not always enhance accuracy.

# *IV. Spatio-Temporal Classification Based on Fractal Dimension*

## *4.1 Introduction*

This chapter presents an application of the embedding theorem to the problem of moving object classification. In this problem, shape feature vectors of several classes of objects are identified. As these objects move through space (or appear to move through space), sequences of views are taken and translated into sequences of feature vectors. The evolutions of the individual components of these feature vectors comprise a set of time series. Depending on how the views are obtained, these time series may reveal an underlying fractal nature to the dynamical systems defined by the evolving feature vectors. It is shown that the embedding theorem may be applied to the highest of the fractal dimensions corresponding to the objects to set a reasonable length for the test sequences presented for classification.

The following section will describe an experimental method of obtaining views of objects. In Section 4.3, a proof is given that a monotonic mapping of a time series yields another time series with the same fractal dimension. Section 4.4 considers an example classification problem using feature vectors consisting of Fourier components of the viewed objects. A nearest neighbor spatio-temporal classifier is described in Section 4.5, and some results of its application to the example classification problem are presented. Section 4.6 discusses strategies for viewing sphere traversals, pointing out that chaotic trajectories may not be necessary for fractal dimension-based classification.

## *4.2 Creating Spatio-Temporal Training Sequences*

Building on earlier work by Rabiner and others (30) (34), Capt Ken Fielding has implemented a Hidden Markov Model spatio-temporal classifier (12, 10, 11, 9). The sequences of views he uses are obtained by traversing (in computer simulation) a portion of the surface of a sphere having at its center a three-dimensional model of the object of interest. The portion of the surface traversed will be called the viewing quadrant, although it doesn't quite fill a quadrant of the sphere. Fielding's technique can be applied to sets of data corresponding to

arbitrary traversals simulating a wide range of possible object motions. Five objects (models of military land vehicles) are positioned at the center of the sphere and provide five classes for recognition. His Hidden Markov Models can be trained on sequences of feature vectors extracted from lengthy training sequences of views. Their performance is then judged using short test sequences of such views which are typically trajectories near, but not entirely overlapping, the training trajectories.

One might wonder how best to assemble training sequences. Perhaps a region-filling curve composed of segments found most typical of actual vehicle movements (derived, for example, from observations made during training exercises) might most accurately represent real-world scenarios. Even if it is practical, however, to train with sequences containing subsequences near every possible test sequence, some critical questions have to be answered. How is "nearness" measured? What minimum length test sequence should be used?

The problem of constructing an optimal training trajectory is not addressed here. Instead, simplifying assumptions are made on the vehicle-viewer orientations to produce training trajectories on the viewing sphere, and the fractal dimensions of the derived feature vector trajectories are then exploited to infer a sufficient lower bound for test sequence lengths.

Suppose, for example, that a training trajectory is chosen which has a known fractal nature. Is it possible to produce a sequence of feature vectors which derives a fractal nature directly from that of the training trajectory? At least for a simple feature extracted from a simple object, the answer, it will be shown, is yes. But first a necessary digression.

### *4.3 Time Series Having Identical Fractal Dimension*

The two time series shown in Figure 29 have the same fractal dimension; a new theorem will be proven in this section to verify this assertion. The reader who chooses not to skip this development may wish to consult Appendix A for some basic concepts and terminology.

Figure 30 shows two disjoint fractal objects (differently warped nested triangles in the upper left and lower right). Barnsley provides a theorem from which it may be deduced that

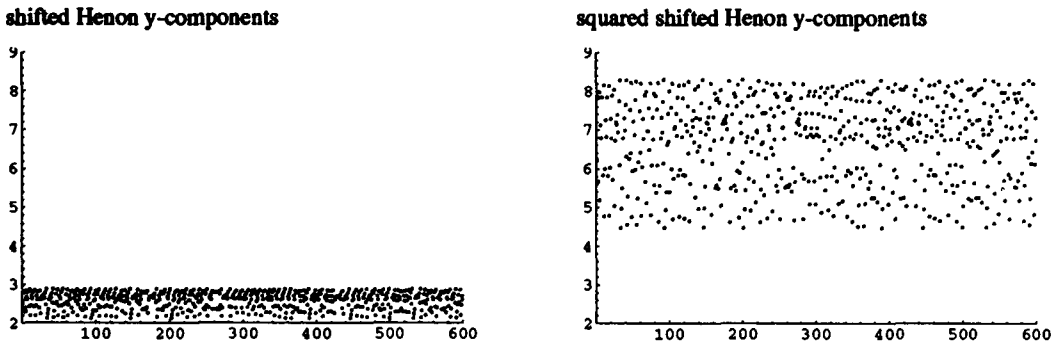



Figure 29. Stretching a Time Series

these objects have the same fractal dimensions. This theorem may also be used to demonstrate that the two time series of Figure 29 have the same fractal dimensions.

In both Figure 29 and Figure 30, the right-side object is the image of the left-side object under a mapping known as a metric equivalence.

*Definition:* Two metric spaces $(X_1, d_1)$ and $(X_2, d_2)$ are *metrically equivalent* if there is a bijective mapping $h : X_1 \to X_2$ and constants $0 < c_1 < c_2 < \infty$ such that for all $x$ and $y$ in $X_1$,

$$c_1 d_1(x, y) \leq d_2(h(x), h(y)) \leq c_2 d_1(x, y)$$

In this context, the mapping $h$ will be called a *metric equivalence.*

A metric equivalence may be thought of as a function which stretches or compresses one space into another, but in such a way that distances between points are neither increased nor decreased unboundedly. The linear mapping $f : [0, 1] \to [0, 2]$, $f(x) = 2x$, is an example of a metric equivalence between the metric spaces $X_1 = [0, 1]$ and $X_2 = [0, 2]$, where both metrics $d_1$ and $d_2$ are assumed Euclidean.

Barnsley provides a proof of the following theorem (2:180).

**Theorem 1.** *Let $(X_1, d_1)$ and $(X_2, d_2)$ be metrically equivalent metric spaces with a metric equivalence $h : X_1 \to X_2$. Suppose a nonempty compact subset $A_1$ of $X_1$ has fractal dimension $D$. Then its image $h(A_1)$ also has fractal dimension $D$.*

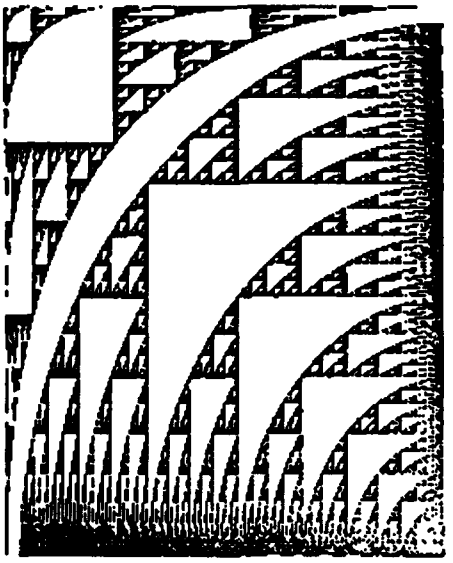


Figure 30. Two Objects in $\Re^2$ with Same Fractal Dimension (2:173)

There is a metric equivalence $h : (2,3) \rightarrow (4,9)$ between a pair of Euclidean spaces which contain the two time series illustrated in Figure 29, namely, $h(x) = x^2$. It is because this metric equivalence extends to the embedding spaces in which the fractal dimensions of the time series are computed that the equivalence of their dimensions may be asserted.

**Theorem 2.** *Suppose $x_i \in (a,b)$ for $i = 1,2,3,\ldots$ and the time series $\{x_i\}$ has fractal dimension $d$. Suppose $g : (a,b) \rightarrow \Re$ is one-to-one and differentiable, and satisfies*

$$0 < \alpha < \left|\frac{dg(z)}{dz}\right| < \beta$$

*for some real constants $\alpha$ and $\beta$ and for all $z \in (a,b)$. Then the time series $\{g(x_i)\}$ also has fractal dimension $d$.*

**Proof:** By the embedding theorem, there is an integer $k > 2d$ such that, in the space $(a,b)^k$ with Euclidean metric, the points $\mathbf{x}_i = (x_{i-k+1}, x_{i-k+2}, \ldots, x_i)^\top$, $i \geq k$, lie on an attractor having fractal dimension $d$. Define a vector function $\mathbf{g} : ((a,b)^k, \text{Euclidean}) \rightarrow$

$([g(a,b)]^k$, Euclidean) by

$$\mathbf{g}\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{bmatrix} g(y_1) \\ g(y_2) \\ \vdots \\ g(y_k) \end{bmatrix}$$

for all $\mathbf{y} = (y_1, y_2, \ldots, y_k)^\top$ in $(a,b)^k$. It will be shown that $\mathbf{g}$ provides an equivalence of the metric spaces $(a,b)^k$ and $[g(a,b)]^k$ (both with assumed Euclidean metrics). Theorem 1 then proves the assertion about $\{g(x_i)\}$.

Let $\mathbf{y} = (y_1, y_2, \ldots, y_k)^\top$ and $\mathbf{z} = (z_1, z_2, \ldots, z_k)^\top$ be any elements of $(a,b)^k$, and let $i \in \{1, 2, \ldots, k\}$. By the mean value theorem of calculus, there is a number $c_i$ in $(y_i, z_i)$ (or in $(z_i, y_i)$ if $z_i < y_i$, or equal to their common value if $y_i = z_i$) such that

$$g(z_i) - g(y_i) = g'(c_i)(z_i - y_i)$$

By hypothesis, $\alpha < |g'(c_i)| < \beta$, and since $|g(z_i) - g(y_i)| = |g'(c_i)||z_i - y_i|$,

$$\alpha|z_i - y_i| \leq |g(z_i) - g(y_i)| \leq \beta|z_i - y_i|$$

$$\alpha^2|z_i - y_i|^2 \leq |g(z_i) - g(y_i)|^2 \leq \beta^2|z_i - y_i|^2$$

This is true for every $i \in \{1, 2, \ldots, k\}$ so

$$\alpha^2 \sum_{i=1}^{k} |z_i - y_i|^2 \leq \sum_{i=1}^{k} |g(z_i) - g(y_i)|^2 \leq \beta^2 \sum_{i=1}^{k} |z_i - y_i|^2$$

$$\alpha \left( \sum_{i=1}^{k} |z_i - y_i|^2 \right)^{1/2} \leq \left( \sum_{i=1}^{k} |g(z_i) - g(y_i)|^2 \right)^{1/2} \leq \beta \left( \sum_{i=1}^{k} |z_i - y_i|^2 \right)^{1/2}$$

$$\alpha\|\mathbf{z} - \mathbf{y}\| \leq \|\mathbf{g}(\mathbf{z}) - \mathbf{g}(\mathbf{y})\| \leq \beta\|\mathbf{z} - \mathbf{y}\|$$

Thus $\mathbf{g}$ is indeed a metric equivalence. ■

Suppose now that the $x$-component of a viewing quadrant traversal forms a time series $x_1, x_2, x_3, \ldots$ with a known fractal dimension. Suppose also that measurements are taken of an object at the center of the viewing sphere from each viewing quadrant location, and those measurements are interpreted for classification purposes as representing a feature of the object. If that feature changes strictly monotonically with increasing $x$ but not at all with other viewing location coordinates, then Theorem 2 implies that the sequence of feature vectors must have the same fractal dimension as the time series $x_1, x_2, x_3, \ldots$. Toward the goal of realizing a feature space trajectory with a known fractal dimension, the next section will consider a novel viewing quadrant training trajectory.

### *4.4 A Lorenz Traversal of the Sphere (An Embedology Perspective)*

Consider a scaled Lorenz attractor confined within 3-space to the viewing quadrant of interest, where the stationary object is centered at the origin and the $x$ and $y$ axes determine the equatorial plane; see Figure 31. Its evolving $x$ components generate a fractal times

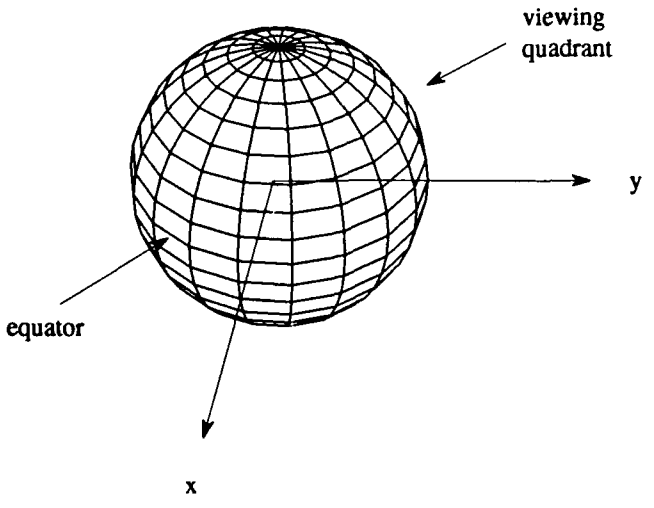



Figure 31. Viewing Sphere

series. Let P denote the projection of the confined attractor onto the viewing quadrant surface (it does not extend to the edges of the surface, and leaves large portions of the viewing

quadrant unexplored). Now suppose that the object of interest at the center of the sphere is a circular cylinder coaxial with the $y$-axis of the coordinate frame. If the cylinder is painted with two longitudinal black and white stripes, with white paint over slightly more than the region observed during traversal, then intensity could serve as a feature which is a monotonic transformation of the $x$-component of the traversal. Figure 32 illustrates how more white becomes visible with increases in viewing position $x$-component, regardless of $y$-component.


Figure 32. A Monotonically Increasing Feature

Thus the trajectory of the intensity feature (in $\Re$) is a time series having the same fractal dimension as the Lorenz attractor.

Such simple objects are of little practical interest, and features other than intensity are often used. Suppose the Lorenz traversal is preserved, but more complex objects are viewed using different features. In his work, Capt Fielding uses features consisting of vectors of 28 dominant two-dimensional spatial Fourier magnitudes associated with the views (12). That is, 28 low frequency pairs are fixed, the spatial Fourier transform is evaluated at each of those pairs, and the magnitudes of those Fourier transforms form 28-dimensional feature vectors. Thus each view is condensed to a 28-tuple of real numbers. A training sequence then consists of a finite number of such 28-tuples, and a test sequence consists of a smaller number of 28-tuples derived from viewing trajectories oriented near the viewing trajectory used during training.

Fourier components do not change monotonically with viewing position. Nevertheless, they do seem to change fairly continuously with viewing position. The five images in Figure 33 (courtesy Ken Fielding) each correspond to the flattened viewing quadrant. Each point within each of these images thus corresponds to a position on the viewing quadrant. The gray-scale intensity at each point represents the value of the fourteenth Fourier component of the object

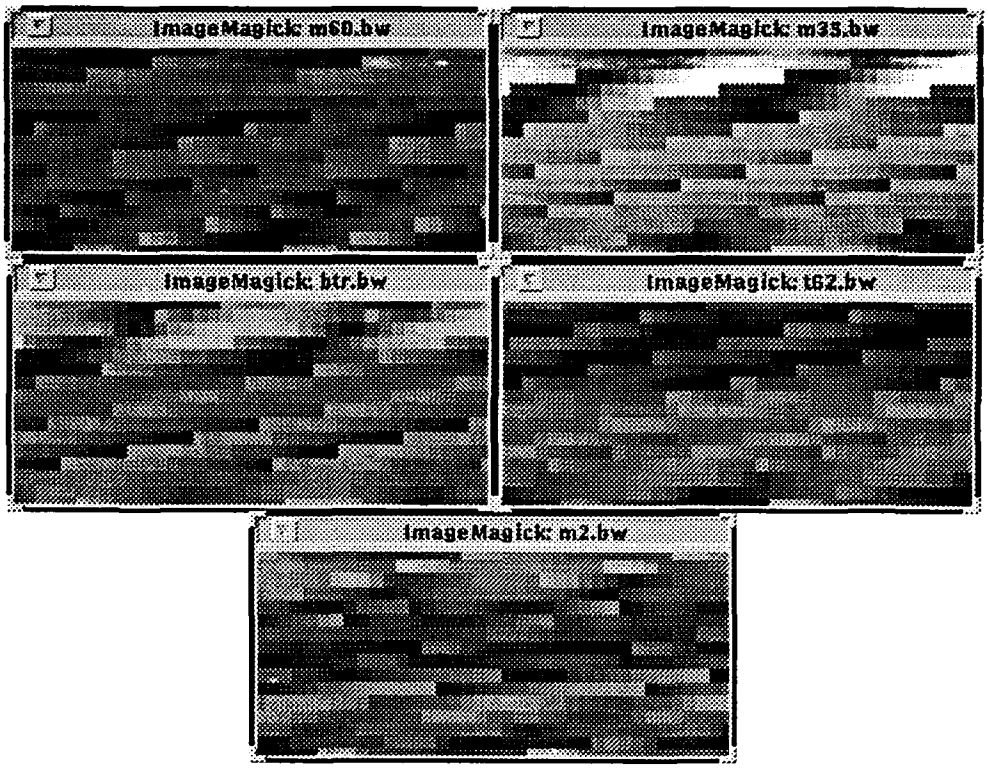



Figure 33. Fourteenth Fourier Components of Five Military Vehicles

at the center of the viewing sphere, as viewed from that point. The objects at the center of the sphere were models of an M60 tank, M35 truck, BTR60 armored personnel carrier, T62 tank, and M2 infantry fighting vehicle, as labeled. The darker is the gray in these images, the higher is the component value (with white being zero). Notice the rarity of sharp discontinuities; perhaps this is due to the complexity of these vehicles, i.e., the large number of parts with both sharp and smooth edges, no single one of which is dominant. In any case, the distributions of fourteenth component values generated from the Lorenz-derived viewing path appear rather continuous. The fourteenth component values seemed typical in this respect to other components examined.

Consider Figure 34, in which the viewing trajectory P is identified. The Lorenz attractor is a compact subset of $\Re^3$ (as is any curve which is a closed bounded subset of $\Re^3$ (1:59)). Now P is the projection of the Lorenz attractor confined to the viewing quadrant, and the mapping

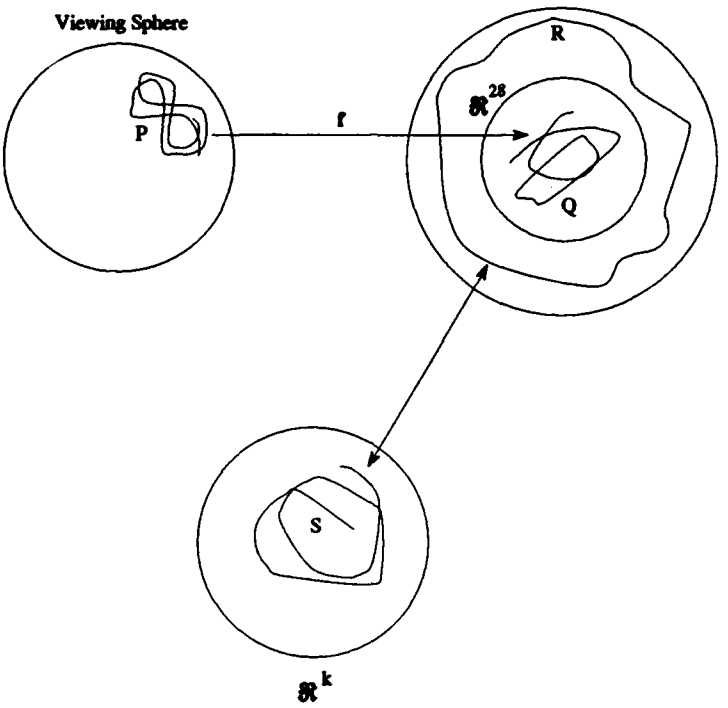



Figure 34. Embedding a Time Series Derived from Views along Trajectory P

which confined the Lorenz attractor is a linear transformation. Both projections and linear transformations are continuous, so the composition mapping with image P is also continuous. Thus P is compact, since continuous images of compact sets are compact (1:82).

Let **f** denote the mapping of P into the 28-tuples of Fourier vectors associated with each viewing position in P. It was argued above that each of the component functions of **f** are continuous (at least, they will be presumed continuous); therefore **f** itself is continuous. Since P is compact, the image Q of P under **f** is a compact subset of $\Re^{28}$. Since all nonempty compact subsets of $\Re^{28}$ have fractal dimensions, Q has a fractal dimension; call it *d* (2:183).

Assume now that the trajectory Q in $\Re^{28}$ is the solution trajectory of a system of differential equations in $\Re^{28}$, or the projection of a solution trajectory R in a higher dimensional space (as the two-dimensional Lorenz curve illustrated in Figure 6 is the projection of a three-dimensional solution trajectory). This requires that Q be continuous; but it has been argued

that the mapping f : $P \rightarrow Q$ is continuous, and since P is continuous, Q also must be continuous (1:79).

The embedding theorem can now be applied to the trajectory Q. Consider any one of the components of Q. A delay coordinate embedding of the time series corresponding to that component into $\Re^k$ yields a diffeomorphic image S of Q (or of R, assuming now that R has fractal dimension $d$), provided only that $k > 2d$.

Thus, any $k$ consecutive values of any one component of Q determine a single point on S. Consider now the relationship between Q and S. For greater generality, suppose Q is projected from a higher dimensional trajectory R. The fundamental theorem of differential equations (18:162) assures that any point on R determines all of R. Since S and R are diffeomorphic, any single point on S not only determines all of S, but its image in R determines all of R. Thus the projected image of the point on S traces the trajectory Q. Therefore *any k consecutive views of the object completely determine the trajectory* Q, which will be called the "object evolution" of the viewed object, to account for its temporal as well as spatial nature.

Consider now the problem of distinguishing one object from another by distinguishing one's evolution from the other's; say, distinguishing an M60 tank's evolution from that of an M35 truck. A priori, there is no reason to believe there is no overlap of evolutions. That is, there may be a point, or even consecutive points, shared by the M60 and M35 evolutions. Let $K$ be the maximum of the embedding dimensions $k_{M60}$ and $k_{M35}$ found sufficient for determining the evolutions of the M60 and M35, respectively. Then provided there are no more than $K - 1$ consecutive shared points on the evolutions, the M60 and M35 may be distinguished simply by matching a test sequence of $K$ 28-tuples (taken from one of the two evolutions) to both evolutions. The evolution which matches identifies the object of the test sequence. If a maximum $\mathcal{M}$ of consecutive points on the evolutions are found to overlap, where $\mathcal{M} > K - 1$, then $\mathcal{M} + 1$ points will suffice for the test sequence.

In fact, if test sequences are only taken from one or the other known evolution, then $\mathcal{M} + 1$ consecutive test sequence points will suffice for identification, regardless of whether $\mathcal{M} > K - 1$. However, given a test sequence of observations which was obtained not

from a partial traversal of P, but perhaps just near P, the task of object classification is more problematic. Intuitively, the object evolution in $\Re^{28}$ which most closely matches the evolution determined by the test sequence, probably corresponds to the object of the test sequence. To maximize the probability of correct identification, however, at least $K$ consecutive views should be included in the test sequence, because $K$ captures something of the "dynamical essence" of the evolutions. Consider Figure 35, in which are depicted fanciful evolutions for

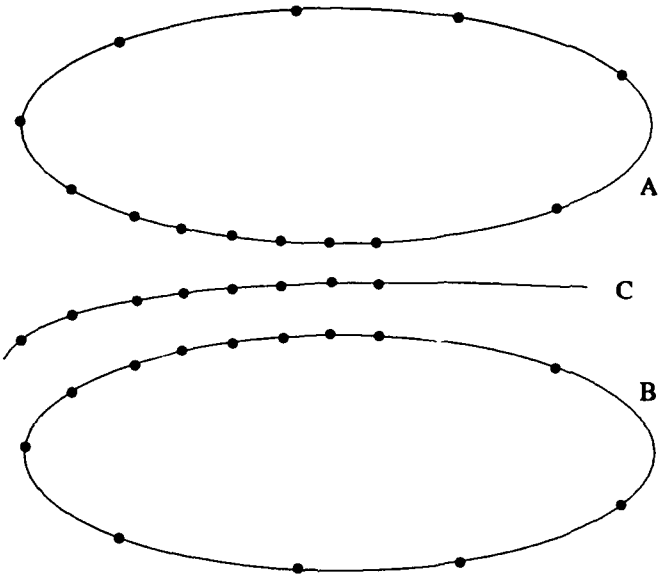



Figure 35. Nonoverlapping Evolutions and a Test Evolution

some objects A and B, and an evolution C corresponding to views of either object A or B which were taken near but not on the viewing sphere traversal common to A and B. Since the evolutions of A and B are disjoint, a single view from anywhere on their common viewing traversal will suffice to classify the test object. However, one, or even more, views taken off the common viewing traversal may not unambiguously identify the object. If it is found, however, that $K = 7$, and seven or more views are taken in determining C, then a correct classification is more likely.

### *4.5 Experimental Application*

The Lorenz traversal just described was applied to obtain training sequences of views of each of five military target classes. After the confined Lorenz attractor was generated, the $(x, y)$ components of its points were projected onto the surface of the sphere, and the resulting path followed for 2000 points (repeated for each of the five classes). At each of the 2000 views, Fourier 28-tuples were extracted. Time series were formed from the 1st, 14th, and 28th components of the resulting sequence of 2000 Fourier vectors. Grassberger and Procaccia analyses of each class revealed nearly linear log-log plots from which fractal dimensions could be estimated. See for example Figure 36, in which the fourteenth Fourier component of the

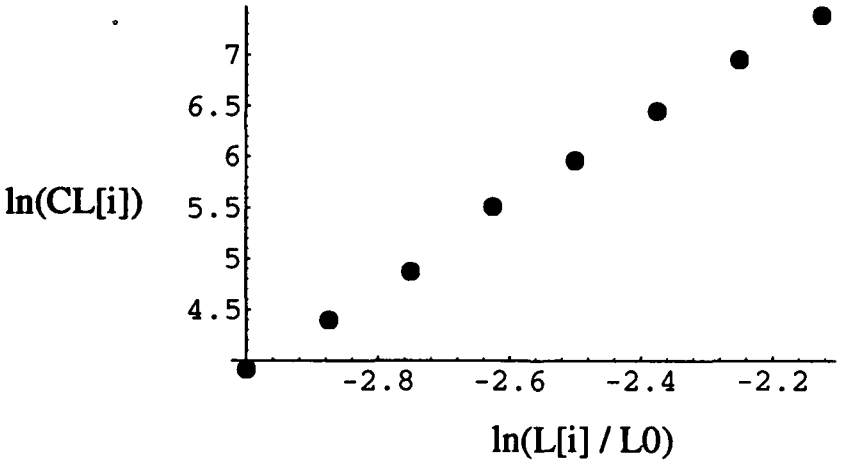



Figure 36. G. and P. Plot for the $14^{th}$ Fourier Component of an M60

M60 sequence was examined. Based on the slope of the line joining the endpoints of this data, the correlation dimension of the generating time series is estimated at about 3.96.

Similar investigations were performed for the 1st and 28th components of the M60 data, and for the 1st, 14th, and 28th components of the remaining four military vehicles. The maximum fractal dimension so obtained (over all 15 values computed) was 5.25; this number was used to infer a minimum test sequence length of $11 > 2 \times 5.25$.

Capt Fielding, in his work with Hidden Markov Models, used training sequences which were not derived from a Lorenz viewing trajectory (12). Typically they were derived from roughly east-to-west (or west-to-east) viewing trajectories, and test sequences of similar "horizontal" orientations were used. Despite the different training sequences, he found in his experiments that test sequences of length eleven were quite adequate for accurate classification.

A nearest neighbor classifier was implemented to determine if test sequences of length 11 were adequate for accurate vehicle classification using the Lorenz-derived training evolutions. Given a test sequence, this technique for determining its class is to match its point-by-point nearness to all sequences of the same length contained in all the training data. The class of the training sequence providing the best fit is declared the most likely class. Consider, for example, hypothetical portions of five training sequences $C^i$ as shown in Figure 37. The

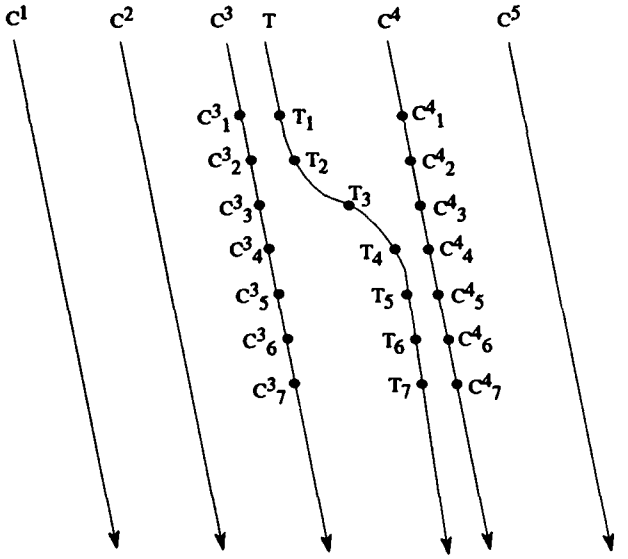



Figure 37. Trajectories in $\Re^{28}$

sampled test sequence in Figure 37 consists of just seven points, and its trajectory through phase space is labeled $T$. The sum of the squared Euclidean distances from the seven points $T_1$ through $T_7$ to the points $C_1^3$ through $C_7^3$, respectively, is greater than the sum of the squared distances from $T_1$ through $T_7$ to the points $C_1^4$ through $C_7^4$, respectively. Indeed, there is no

sequence of seven points in any class which is closer to the test sequence than the seven points $C_1^4$ through $C_7^4$. The test sequence is therefore identified as class four.

An equivalent statement of the classification procedure may be found by imagining the test sequence a single point T in $7 \times 28 = 196$-space. That is, concatenate the 28 components of each of the 7 points into a single vector T of dimension 196. Starting with each point (prior to the final six) in all the training classes, form 196-tuples from the concatenated components of the starting point and the six points following it. The class of the point in 196-space nearest T in Euclidean distance is declared the class to which T belongs. This is the approach taken in developing the nearest neighbor algorithm used in this research.

Test sequences were obtained near the Lorenz viewing trajectory, but not on it, by solving the Lorenz equations using a different initial condition than that used to generate the training trajectory. Figure 38 shows the projected solution of the Lorenz equations when an

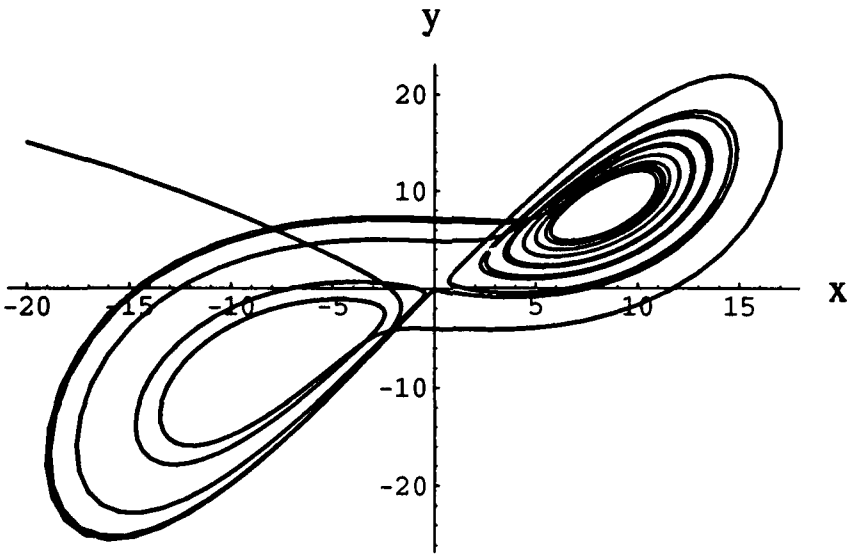



Figure 38. Obtaining Lorenz Test Sequences

initial condition of $(x_0 = -20, y_0 = 15, z_0 = 15)$ was used. Ignoring the initial point, the next 20 points on this trajectory were used to construct test sequences of lengths 4, 8, and 11 (this allowed 17 sequences of length 4, 13 of length 8, and 10 of length 11). The $20^{th}$

point is the first point after the knee of the initial portion of the trajectory; it is located at about $(x = -2, y = -4)$. Compare this trajectory to the trajectory used to derive the training data. This is illustrated in Figure 6, where the initial condition used to determine the trajectory was $(x_0 = 5, y_0 = 5, z_0 = 5)$. Using the test sequences with all five vehicles, classification accuracy was 100% for sequences of lengths 11 and 8, and 98% for sequences of length 4. Thus sequences of length 11 proved sufficiently long to classify these particular test sequences perfectly; but so did sequences of length 8, and even sequences of length 4 classified quite well.

There are several possible explanations for these results. First, the military vehicles may be so different that their training trajectory-derived evolutions in $\Re^{28}$ overlap very little, or not at all (for example, as in Figure 35). If this is the case, very short test sequences will likely suffice for accurate classification. Second, there could be considerable overlap of evolutions, but the test sequences applied didn't fall near the regions of overlap. Third, the embedding theorem gives a sufficient, not necessary, sequence length to completely determine an object's evolution. For example, eleven training sequence views of an M60 are *sufficient* to completely determine the M60's evolution in $\Re^{28}$; a number less than eleven may also suffice.

Test sequences were derived from Lorenz solutions using a couple of other initial conditions, with similar results. In both cases, using test sequences of length eleven yielded 100% classification accuracy, but often sequences of shorter length did the same. Perhaps hundreds of experiments with different test sequences could yield statistics which would tend to confirm or deny that test sequences of length eleven are sufficient for accurate classification. The test sequences used, however, could not possibly be comprehensive – there are uncountably many possible test sequences – so they would have to be chosen with some application or geometric criterion in mind. It should be considered, too, that "rogue" test sequences can be contrived such that a short test length classification will be more accurate than ones of longer length.

Figure 39 illustrates the uncertainty inherent in spatio-temporal classification. The curve marked $T_{M60}$ represents a portion of a training trajectory in $\Re^{28}$ corresponding to a particular viewed object M60. The points $A$, $B$, and $C$ represent points in $\Re^{28}$ derived from consecutive

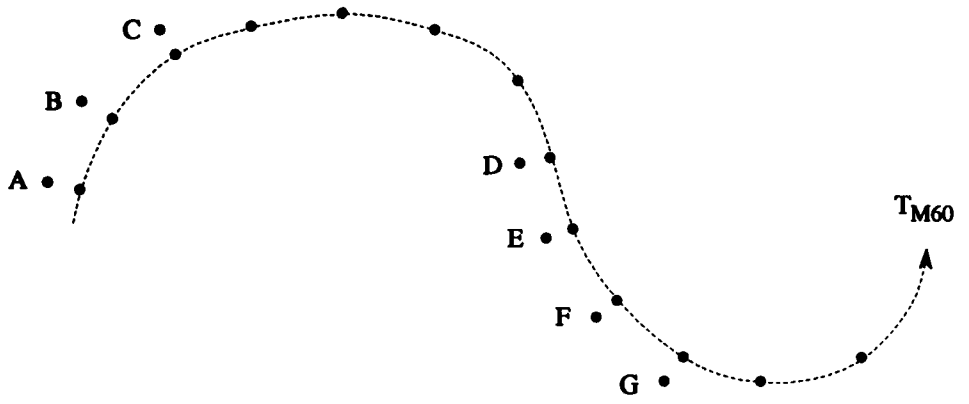



Figure 39. Portions of Trajectories in $\Re^{28}$

testing views of the object taken near, but not on, the training trajectory; so do the points $D, E, F$, and $G$, obtained during a separate test. Suppose that all the points $A$ through $G$ are at the same small Euclidean distance from points on $T_{M60}$, and that none of the other testing points are as close. How likely is it that the object viewed during the test traversal which yielded the points $A, B$, and $C$ is an M60? Less likely, intuitively, than the object viewed during the test traversal which yielded the points $D, E, F$, and $G$, because the latter testing trajectory tracked the training trajectory over a longer period of time. Similarly, greater confidence would attach to five consecutive close points than to four. The significance of the number eleven is that if eleven consecutive testing points coincide exactly with eleven consecutive points on $T_{M60}$, and the testing sequence was obtained following the same viewing quadrant traversal as the one which yielded $T_{M60}$, then the testing trajectory and $T_{M60}$ must be identical. For testing trajectories not directly on $T_{M60}$, twelve or more consecutive near points may reasonably be interpreted as stronger evidence of an M60 than eleven consecutive near points. Nevertheless, lacking specification of allowable test sequences, the embedology-derived number eleven may be a reasonable compromise test sequence length to use in practice.

### *4.6 Discussion of Viewing Sphere Traversal Strategies*

The foregoing use of a Lorenz viewing sphere traversal is not meant to imply advantage for that strategy as opposed to any other strategy. Any traversal which gives feature vectors as a continuous function of viewing position will yield a feature space trajectory which has a fractal dimension (this is because any traversal of the viewing sphere is a compact subset of $\Re^3$). As a finite-length continuous curve in feature space, however, its fractal dimension is one. On the other hand, the fractal dimension of a discrete sampling of points on the curve may well be greater than one. Consider, for example, a finite-length Lorenz trajectory in $\Re^3$. As a curve in $\Re^3$, it has a fractal dimension of one. However, a discrete sampling of points on the trajectory will exhibit self-similarity in space, and a fractal dimension-determining algorithm will yield values approaching 2.05 for the set of points (as the length of the Lorenz curve becomes large).

An experiment conducted with another viewing sphere traversal strategy, a sort of random walk scan of the viewing quadrant, suggests that a fractal structure to the viewing traversal may simplify the determination of fractal dimensions of feature space solution trajectories. Perhaps the training evolutions "inherit" some of the fractal nature of a fractal viewing trajectory. Figure 40 illustrates a typical Grassberger and Procaccia plot associated with the $14^{th}$ Fourier component of a randomly-viewed M60 tank. The relative lack of collinearity of the data points (compare Figure 36) weakens confidence in any fractal dimension derived from the "common" slope of the line segments joining them (17).

On the other hand, fractal structure in the viewing traversal may have the effect of increasing the number of points required for determination of the training trajectory. Consider again the intensity feature for the painted cylinder discussed in Section 4.4. If a second cylinder is introduced for a classification problem, identical to the first except that the second cylinder's white paint is brighter, then fractal dimension-based classification using the Lorenz traversal will require five views of a test object (i.e., one of the two cylinders) to make a classification. This is because the fractal dimension of the Lorenz attractor is about 2.05, and five is the smallest integer greater than twice this dimension. On the other hand, if a simple

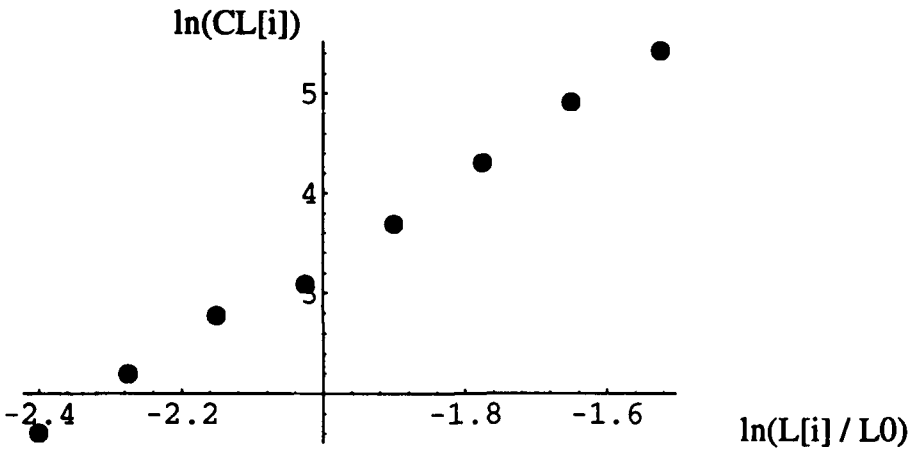



Figure 40. G. and P. Plot for an M60 Viewed along a Random Trajectory

east-to-west, west-to-east raster traversal had been used instead of the Lorenz, then fractal dimension-based classification will require only three views of the test object. This is because the fractal dimension of the sinusoidal intensity features of both objects is one.

Notice that both methods of traversal can be spoofed by "unanticipated" test sequences. The raster traversal method, for instance, could easily be fooled by a north-to-south test sequence. This points out what is perhaps the biggest obstacle to constructing successful spatio-temporal classifiers: anticipating likely test sequences and accounting for them in training trajectories.

### *4.7 Conclusion*

This chapter has demonstrated that the goal of moving object recognition might be facilitated if the motion (or apparent motion) of the objects is chosen carefully and appropriate features are utilized. A fractal dimension of object evolutions can be obtained and exploited if the views of the given objects form a compact subset of the viewing space, and the features used

vary continuously with viewing position. The fractal dimension is exploited via the embedding theorem to determine a test sequence length sufficient for complete object determination.

A nearest neighbor spatio-temporal classifier was implemented to test the theoretically determined sufficient length of test sequence against Fourier data obtained from models of five military objects. These objects were observed while following a Lorenz-derived traversal of a viewing sphere. The results revealed that indeed the theoretically sufficient test sequence length was sufficient; but much shorter lengths also sufficed.

# *V. Conclusions*

The Deterministic Versus Stochastic (DVS) algorithm developed by Martin Casdagli uses the local linear prediction technique described by Doyne Farmer to help decide whether a given time series represents low dimensional (deterministic) chaotic dynamics, or is high dimensional (or stochastic) in origin. This algorithm has been modified here to produce two new prediction methodologies, each of which selectively uses embedding space nearest neighbors. Both have been shown advantageously applicable to prediction of noisy time series (such as experimentally measured blood oxygen concentration data, and certain financial data).

The first algorithm, pruned outliers prediction, excludes from the DVS-determined optimal number of neighbors nearest the prediction point, those of them which lie farthest from the linear prediction hyperplane. A new hyperplane is calculated based on the remaining neighbors, and the value assumed on this hyperplane at the prediction point becomes the predicted value.

The second algorithm, overlap prediction, endeavors to combine the benefits of using one short and one longer time delay. Prediction is based on the shorter of the two time delays, but intervals of time are allowed to be represented in the utilized set of nearest neighbors only if they play a role in both short term and long term prediction.

In each new algorithm, neighbors which are considered prediction-relevant are retained for local linear prediction, while those which are considered likely to represent noise are ignored. These algorithms may in this sense be considered to employ embedding space filtrations of the time series. For many time series, it was found rather easy to improve on unfiltered local linear prediction with one or both of the new algorithms. For other time series, prediction improvement was more difficult. It was argued that prediction improvement difficulty is indicative of stochastic data, independently of the direct results of the DVS algorithm.

This research has shown that the local linear technique provides *perfect* predictions in cases where phase space trajectories are concentric circles, and where they are locally parallel.

This helps to explain why attempts to use higher-order polynomial regression prediction sometimes provide little benefit over linear regression prediction (8:846).

Another new result gives an invariance condition for the fractal dimension of a time series. Specifically, a theorem is proven which shows that the fractal dimension of a time series does not change if a differentiable mapping which is strictly monotonic on the series' range of values is applied to it.

It has also been shown that the problem of moving object classification may be analyzed in terms of embedded time series, in the sense that embedology can supply a reasonable length of test sequence for accurate classification. Sequentially recorded feature vectors of a moving object form a training trajectory in feature space. Each of the sequences of feature vector components is a time series, and under certain conditions, each of these time series will have approximately the same fractal dimension. The embedding theorem may be applied to this fractal dimension to establish a number of observations sufficient to determine the feature space trajectory of the object. It was argued that this number is a reasonable test sequence length for use in object classification. Experiments with data corresponding to five military vehicles (observed following a projected Lorenz trajectory on a viewing sphere) showed that this length was indeed adequate.

## *Appendix A. Fractal Dynamics*

In his book *Fractals Everywhere*, Michael Barnsley provides a precise definition of chaos and a proof that certain dynamical systems are chaotic. He also shows how calculated trajectories can maintain their fractal quality even in the presence of computational errors (2). His development is based on the theory of iterated function systems, which is briefly summarized in this appendix. Although Barnsley's book is remarkably self-sufficient, a good first course in mathematical analysis is helpful. Lang's book *Analysis I* is especially recommended (19).

For brevity, this Appendix omits certain definitions and proofs when it is felt that these omissions will not hinder a broad understanding of discrete chaotic dynamical systems. Barnsley's book is the appropriate source for all details.

A *contraction mapping* is a function $f : X \to X$ on a metric space $(X, d)$ (that is, a set $X$ with distance metric $d : X \times X \to \Re$) for which there is a constant $0 \leq s < 1$ (called a *contractivity factor* for $f$) such that for all $x$ and $y$ in $X$,

$$d(f(x), f(y)) \leq s \cdot d(x, y)$$

A *hyperbolic iterated function system* (IFS) consists of a complete metric space $X$ together with a finite set of contraction mappings $w_n : X \to X$, with respective contractivity factors $s_n$, for $n = 1, 2, \ldots, N$. The notation for this IFS is $\{X; w_n, n = 1, 2, \ldots, N\}$. Its contractivity factor is $s = \text{Max}\{s_n : n = 1, 2, \ldots, N\}$. The word *hyperbolic* means that each contractivity factor $s_n$ satisfies $0 \leq s_n < 1$; unless the context clearly indicates otherwise, an IFS is understood to be hyperbolic.

The closed unit interval $[0, 1]$ in $\Re$, with the distance between two points being the absolute value of their difference, is an example of a complete metric space. The function $w_1(x) = x/3$ is an example of a contraction mapping on $X$, with contractivity factor $1/3$; so

is the function $w_2(x) = (x+2)/3$. Thus $\{[0,1]; w_1, w_2\}$ is an IFS, with contractivity factor $1/3$.

Let $\{X; w_n, n = 1, 2, \ldots, N\}$ be an IFS with contractivity factor $s$ on the complete metric space $(X, d)$. There is a derived metric space $\mathcal{H}(X)$ whose elements are nonempty compact subsets of $X$ and whose metric is the Hausdorff distance (denoted $h(d)$ or sometimes simply $h$) between such subsets. With this metric, $\mathcal{H}(X)$ is also complete, so the Contraction Mapping Theorem applies to it. The upshot is that the transformation $W : \mathcal{H}(X) \to \mathcal{H}(X)$ defined by

$$W(B) = \bigcup_{n=1}^{N} w_n(B)$$

for all $B \in \mathcal{H}(X)$ is a contraction mapping on the complete metric space $(\mathcal{H}(X), h(d))$ with contractivity factor $s$. That is,

$$h(W(B), W(C)) \leq s \cdot h(B, C)$$

for all $B, C \in \mathcal{H}(X)$. Its unique fixed point, $A \in \mathcal{H}(X)$, obeys

$$A = W(A) = \bigcup_{n=1}^{N} w_n(A)$$

and is given by $A = \lim_{n \to \infty} W^{\circ n}(B)$ for any $B \in \mathcal{H}(X)$, where $W^{\circ n}(B)$ denotes $n$ iterations of the mapping $W$.

The fixed point $A \in \mathcal{H}(X)$ is called the *attractor* of the IFS.

Recall the classical Cantor set $\mathcal{C}$ created by successive deletions of middle thirds of closed intervals, beginning with the closed unit interval. The set $\mathcal{C}$ is the attractor of the IFS $\{[0,1]; w_1(x) = \frac{1}{3}x, w_2(x) = \frac{1}{3}x + \frac{2}{3}\}$ just described. This IFS is *totally disconnected*; since its maps $w_1$ and $w_2$ are injective, this means that $\mathcal{C}$ is the disjoint union of the images of itself under $w_1$ and $w_2$. That is, $\mathcal{C} = W(\mathcal{C}) = w_1(\mathcal{C}) \cup w_2(\mathcal{C})$ disjointly. Equivalently (in the case of subsets of $\Re$), $\mathcal{C}$ contains no intervals (6:37).

Notice that for a general IFS, an infinite composition of mappings $w_{\sigma_1} \circ w_{\sigma_2} \circ w_{\sigma_3} \circ \cdots (y)$ specifies a unique point on the attractor of the IFS, where the $\sigma_i$ are taken from the set $\{1, 2, \ldots, N\}$ of indices of the contraction mappings and where $y$ is any nonempty compact subset of $X$. For example, the point $1/3$ in $\mathcal{C}$ can be written $w_1 \circ w_2 \circ w_2 \circ w_2 \circ \cdots (0)$. This fact allows a natural correspondence between the attractor $A$ of the IFS and a complete, compact metric space called *code space* whose elements are infinite sequences of symbols taken from $\{1, 2, \ldots, N\}$. Code space is denoted $(\Sigma, d_c)$ (or sometimes simply $\Sigma$) where $d_c$ is a metric defined by

$$d_c(\omega, \sigma) = \sum_{n=1}^{\infty} \frac{|\omega_n - \sigma_n|}{(N+1)^n}$$

for all $\omega, \sigma \in \Sigma$.

The natural correspondence alluded to is in fact a continuous transformation $\phi : \Sigma \to A$ given by $\phi(\sigma) = w_{\sigma_1} \circ w_{\sigma_2} \circ w_{\sigma_3} \circ \cdots (y)$ where $\sigma = \sigma_1 \sigma_2 \sigma_3 \cdots$ and $y$ is any element of $\mathcal{H}(X)$. For the Cantor set example, $\phi(1222\cdots) = 1/3$. The transformation $\phi$ is always surjective but it is not necessarily injective. The IFS is totally disconnected iff $\phi$ is injective.

There is another metric on $\Sigma$ which allows one to picture code space as a subset of the unit interval. In particular, let $d_2 : \Sigma \times \Sigma \to \Re$ be defined by

$$d_2(\omega, \sigma) = \left| \sum_{n=1}^{\infty} \frac{\omega_n - \sigma_n}{(N+1)^n} \right|$$

Then $d_2$ is a metric on $\Sigma$, and $(\Sigma, d_c)$ and $(\Sigma, d_2)$ are equivalent metric spaces. This means that points that are close in $(\Sigma, d_c)$ are also close in $(\Sigma, d_2)$, and that there is no infinite stretching or compression when going from $(\Sigma, d_c)$ to $(\Sigma, d_2)$ (or from $(\Sigma, d_2)$ to $(\Sigma, d_c)$). The $d_2$ metric on $\Sigma$ is precisely the absolute value of the difference between points in the unit interval whose $(N+1)$-ary fractional expansions contain no zeroes. For example, with $N = 2$, $1/2$ can be represented in the unit interval as $.111\cdots$ where the leading period denotes the ternary point; $11/18$ can be represented $.12111\cdots$; and $|.111\cdots - .12111\cdots| = 1/9$.

The concept of dynamical system can now be defined, and subsequently the notion of a chaotic dynamical system.

A *dynamical system* $\{X; f\}$ is a mapping $f : X \to X$ on a metric space $X$. The *orbit* of a point $x \in X$ is the sequence $\{f^{\circ n}(x)\}_{n=0}^{\infty}$.

Examples of dynamical system abound. Two interesting examples of dynamical systems on the unit interval $[0, 1]$ are given in Figure 41.

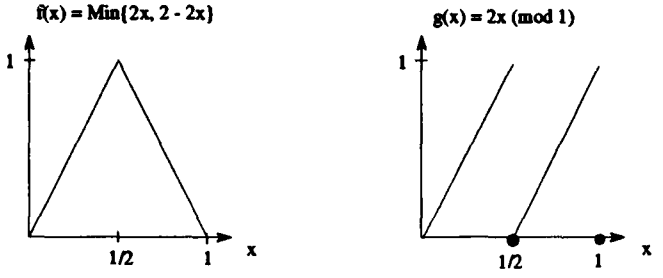



Figure 41. Example Dynamical Systems on the Unit Interval

Iterated function systems admit especially interesting types of dynamical systems. Let $\{X; w_n, n = 1, 2, \ldots, N\}$ be a totally disconnected IFS with attractor $A$. Then $A$ is the union of the $w_n(A)$, where for all $i \neq j$, $w_i(A) \cap w_j(A) = \emptyset$. The associated *shift transformation* on $A$ is the transformation $S : A \to A$ defined by

$$S(a) = w_n^{-1}(a)$$

for $a \in w_n(A)$, where $w_n$ is viewed as a transformation from $A$ onto $w_n(A)$. The dynamical system $\{A; S\}$ is called the *shift dynamical system* associated with the IFS.

Using the shift transformation of a totally disconnected IFS, it is straightforward to trace the orbit of a given point of $A$. A similar procedure can be used to trace orbits when the IFS is not totally disconnected, i.e., when the code space mapping $\phi : \Sigma \to A$ is not injective. For clarity, attention is here restricted to IFS's with only two contraction maps. Let $\{X; w_1, w_2\}$ be an IFS with attractor $A$. Assume that both $w_1$ and $w_2$ are injective. A sequence of points $\{x_n\}_{n=0}^{\infty}$ in $A$ is called an orbit of the *random shift dynamical system* associated with the IFS

if for each $n \in \{0, 1, 2, \ldots\}$,

$$x_{n+1} = \begin{cases} w_1^{-1}(x_n) \text{ when } x_n \in w_1(A), \text{ and } x_n \notin w_1(A) \cap w_2(A), \\ w_2^{-1}(x_n) \text{ when } x_n \in w_2(A), \text{ and } x_n \notin w_1(A) \cap w_2(A), \\ \text{one of } \{w_1^{-1}(x_n), w_2^{-1}(x_n)\}, \text{ when } x_n \in w_1(A) \cap w_2(A) \end{cases}$$

The notation $x_{n+1} = S(x_n)$ is adopted, although there may be no well-defined transformation $S : A \to A$ which makes this true. The pair $\{A; S\}$ is called the *random shift dynamical system* associated with the IFS.

An example of an orbit from a random shift dynamical system is provided in Figure 42. The applicable (overlapping) IFS is $\{[0, 1]; w_1(x) = \frac{1}{2}x, w_2(x) = \frac{3}{4}x + \frac{1}{4}\}$ with respective

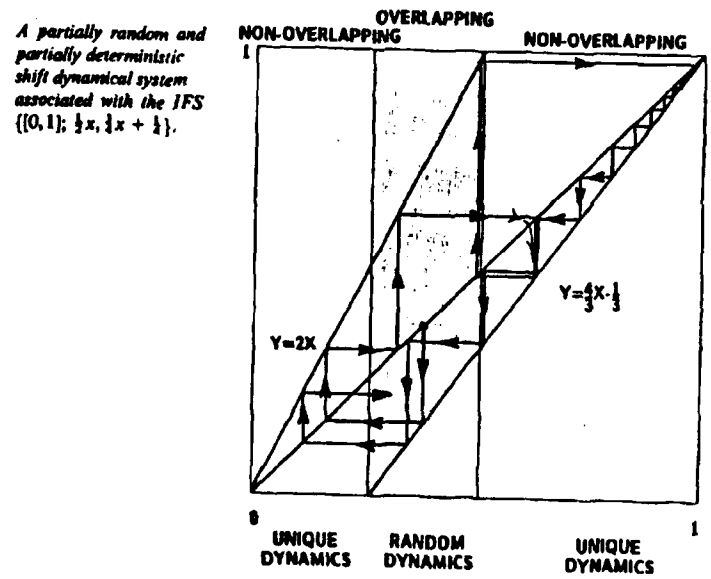



Figure 42. An Example Orbit (2:154)

inverse maps $2x$ and $\frac{4}{3}x - \frac{1}{3}$. Notice that in the overlapping region, the choice of inverse map $w_1^{-1}$ or $w_2^{-1}$ is random.

A deterministic dynamical system can be constructed which acts on a higher dimensional space and whose projection into the original space $X$ yields the random shift dynamics just described. It turns out that the space $X \times \Sigma$ is a complete metric space, with metric defined as the maximum of the pair of metrics $d_x$ and $d_c$ acting on $X \times X$ and $\Sigma \times \Sigma$ respectively. The *lifted* IFS associated with the IFS $\{X : w_1, w_2\}$ is the IFS $\{X \times \Sigma; \tilde{w}_1, \tilde{w}_2\}$ where $\Sigma$ is the code space on two symbols $\{1, 2\}$ and

$$\tilde{w}_1(x, \sigma) = (w_1(x), 1\sigma) \text{ for all } (x, \sigma) \in X \times \Sigma$$
$$\tilde{w}_2(x, \sigma) = (w_2(x), 2\sigma) \text{ for all } (x, \sigma) \in X \times \Sigma$$

The attractor $\tilde{A}$ of the lifted IFS is the graph of the code space map $\phi$; that is, $\tilde{A} = \{(\phi(\sigma), \sigma) : \sigma \in \Sigma\}$. Its projection into the original space $X$ is simply the attractor $A$ of the original IFS. Furthermore, $\tilde{A}$ is totally disconnected in the sense that the projection map from $\tilde{A}$ into $\Sigma$ is one-to-one on $\Sigma$. Figure 43 shows a lifting of the the attractor $A$ of the IFS $\{[0, 1]; w_1(x) = \frac{1}{2}x, w_2(x) = \frac{3}{4}x + \frac{1}{4}\}$. If the maps $w_1$ and $w_2$ in the original IFS are injective, then the lifted IFS is totally disconnected. In this case, the shift dynamical system $\{\tilde{A}, \tilde{S}\}$ associated with the lifted IFS is called the *lifted shift dynamical system* associated with the IFS. The action of $\tilde{S}$ on $\tilde{A}$ is given by

$$\tilde{S}(x, \sigma) = (w_{\sigma_1}^{-1}(x), \sigma_2\sigma_3\sigma_4 \cdots)$$

for all $(x, \sigma) = (x, \sigma_1\sigma_2\sigma_3 \cdots)$ in $\tilde{A}$.

The following "Shadow Theorem" reveals that the orbits of a large class of (possibly overlapping) IFS's can be viewed as projections of orbits of totally disconnected IFS's.

The Shadow Theorem: Let $\{X : w_1, w_2\}$ be an IFS with injective transformations $w_1$ and $w_2$ and attractor $A$. Let $\{x_n\}_{n=0}^{\infty}$ be any orbit of the associated random shift dynamical

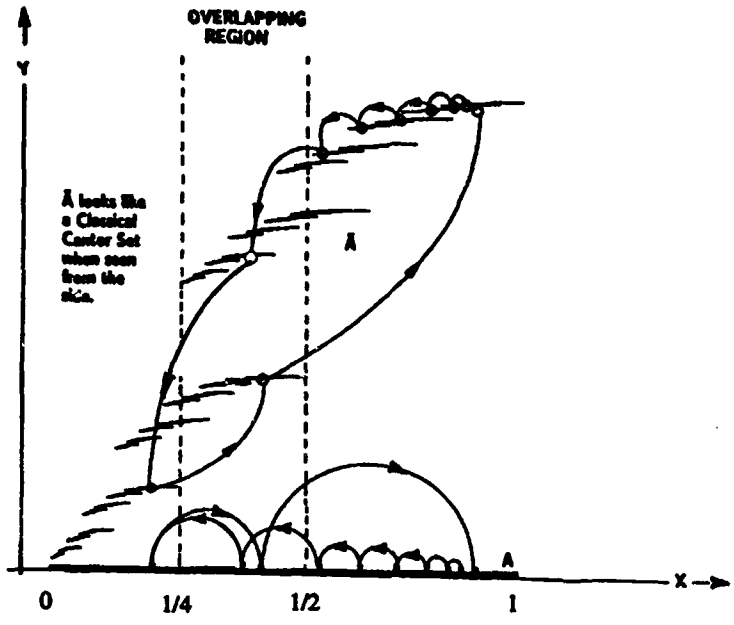



Figure 43. A Lifted Attractor (derived from (2:158))

system $\{A; S\}$. Then there is an orbit $\{\tilde{x}_n\}_{n=0}^{\infty}$ of the lifted shift dynamical system $\{\tilde{A}; \tilde{S}\}$ such that the first component of $\tilde{x}_n$ is $x_n$ for all $n$.

Figure 43 also illustrates the shadow theorem.

Even if one were fortunate enough to be able to identify a point $x_0$ on an attractor with perfect precision (ie, to know perfectly $\sigma \in \Sigma$ such that $\phi(\sigma) = x_0$), he or she probably wouldn't be able to calculate its orbit exactly at each step. That is, one wouldn't be able to calculate exactly the points $x_1 = S(x_0), x_2 = S(x_1), \ldots$. Fortunately, there is a "Shadowing Theorem" which reveals that, regardless of how many small errors are made, there is an exact orbit which lies at every step within a small distance of the errorful one.

The Shadowing Theorem: Let $\{X; w_1, w_2, \ldots, w_N\}$ be an IFS of contractivity $s$, where $0 < s < 1$. Let $A$ denote the attractor of the IFS and suppose that each of the transformations $w_n : A \to A$ is injective. Let $\{A; S\}$ denote the associated shift dynamical system in the case that the IFS is totally disconnected; otherwise let $\{A; S\}$ denote the associated random

shift dynamical system. Let $\{\tilde{x}_n\}_{n=0}^{\infty} \subset A$ be an approximate orbit of $S$, such that

$$d(\tilde{x}_{n+1}, S(\tilde{x}_n)) \leq \theta \text{ for all } n = 0, 1, 2, 3, \ldots$$

for some fixed constant $\theta$ with $0 \leq \theta \leq \text{diam}(A)$. Then there is an exact orbit $\{x_n = S^{\circ n}(x_0)\}_{n=0}^{\infty}$ for some $x_0 \in A$, such that

$$d(\tilde{x}_{n+1}, x_{n+1}) \leq \frac{s\theta}{(1-s)} \text{ for all } n = 0, 1, 2, 3, \ldots$$

Figure 44 shows an approximate orbit of the point $\tilde{x}_0$ on an attractor $A$ (a Sierpinski

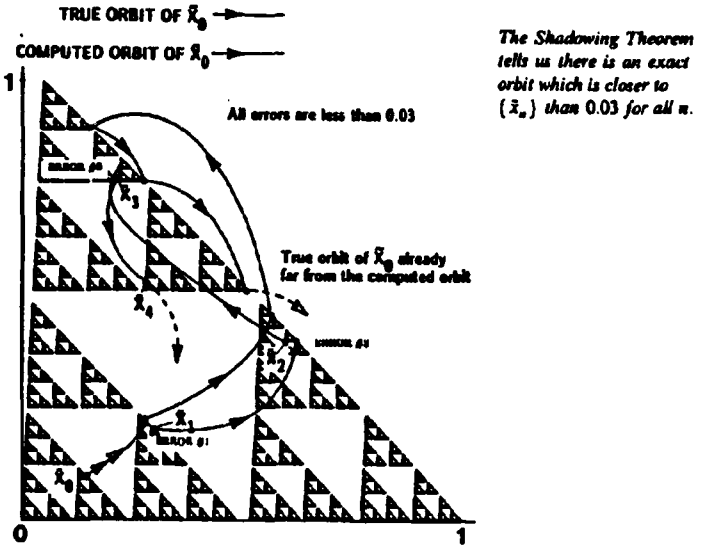



Figure 44. The Effect of Computational Errors (2:163)

triangle). The shadowing theorem assures that there is a point $x_0$ on $A$ whose exact orbit remains less than or equal to $s\theta/(1-s)$ at each pair of points on the orbits.

Most of the concepts necessary to define the phrase "chaotic dynamical system" have now been presented. Only a few more definitions are needed.

Let $(X, d)$ be a metric space. A subset $B \subset X$ is said to be *dense* in $X$ if the closure of $B$ equals $X$. A sequence $\{x_n\}_{n=0}^{\infty}$ of points in $X$ is said to be dense in $X$ if, for each point $a \in X$, there is a subsequence $\{x_{\sigma_n}\}_{n=0}^{\infty}$ which converges to $a$. In particular, an orbit $\{x_n\}_{n=0}^{\infty}$ of a dynamical system $\{X; f\}$ is said to be dense in $X$ if the sequence $\{x_n\}_{n=0}^{\infty}$ is dense in $X$.

A dynamical system $\{X; f\}$ is *transitive* if, whenever $\mathcal{U}$ and $\mathcal{V}$ are open subsets of the metric space $(X, d)$, there exists a finite integer $n$ such that

$$\mathcal{U} \cap f^{\circ n}(\mathcal{V}) \neq \emptyset$$

The dynamical system $\{[0, 1]; f(x) = \text{Min}\{2x, 2 - 2x\}\}$ depicted in Figure 41 is transitive.

The dynamical system $\{X; f\}$ is *sensitive to initial conditions* if there exists $\delta > 0$ such that, for any $x \in X$ and any closed ball $B(x, \epsilon)$ with radius $\epsilon > 0$ there is $y \in B(x, \epsilon)$ and an integer $n \geq 0$ such that $d(f^{\circ n}(x), f^{\circ n}(y)) > \delta$.

This means roughly that orbits which begin close together get pushed apart by the action of the dynamical system. The dynamical system $\{[0, 1]; g(x) = 2x(\text{mod}1)\}$ depicted in Figure 41 is sensitive to initial conditions.

A dynamical system $\{X; f\}$ is now defined *chaotic* if it is transitive, sensitive to initial conditions, and the set of periodic orbits of $f$ is dense in $X$.

The following theorem provides a wealth of chaotic dynamical systems.

Theorem: The shift dynamical system associated with a totally disconnected IFS of two or more transformations is chaotic.

In particular, the shift dynamical system associated with the classical Cantor set $\mathcal{C}$ is chaotic. This means that if the points on $\mathcal{C}$ are "backed out" using the inverses of the maps which provided convergence to them, the resulting set of orbits is chaotic.

## *Appendix B. Fractal Interpolation Functions*

Consider the set of points

$$\{(x_0, F_0), (x_1, F_1), \ldots, (x_N, F_N)\}$$

in $\Re^2$, where $x_0 < x_1 < \ldots < x_N$. There are a number of ways these points can be interpolated with continuous functions on the interval $[x_0, x_N]$; the most familiar of these interpolate using differentiable functions. There is, however, a family of functions which are not necessarily differentiable and which also interpolate these points. These functions are derived from mappings called shear transformations (2:214). Consider for example the three points on the left of Figure 45. An interpolating function is sought which joins these points

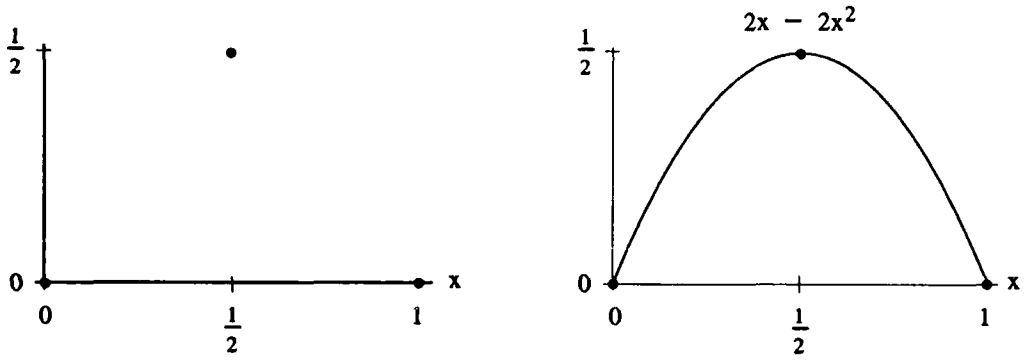



Figure 45. A Parabolic Interpolation of Three Points

$(0,0), (0.5, 0.5)$, and $(1,0)$. One such function, a (differentiable) quadratic polynomial, is illustrated on the right of Figure 45. It is also possible to produce curves in $\Re^2$ which are the attractors of iterated function systems (IFS's; see Appendix A) and which, viewed as real-valued functions on $[x_0, x_N]$, interpolate the points $(0,0), (0.5, 0.5)$, and $(1,0)$. The linear splines interpolation illustrated on the left of Figure 46 is the attractor of the IFS consisting of

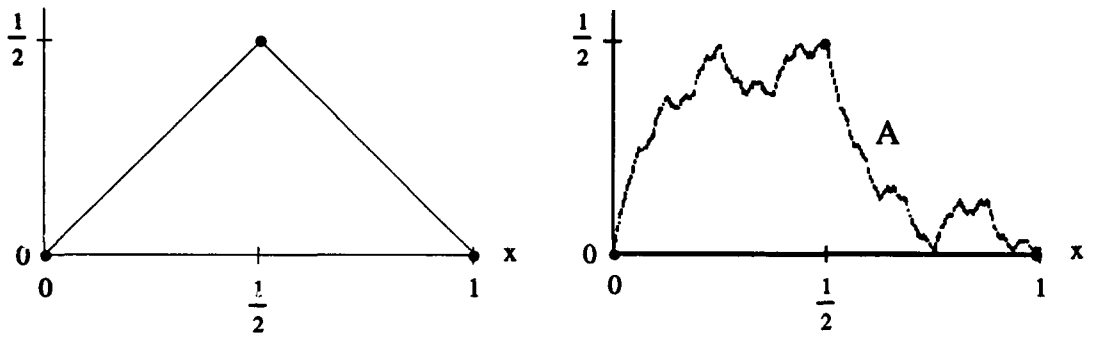



Figure 46. Two Fractal Interpolation Functions (2:223)

the shear transformations

$$f_1\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}, \qquad f_2\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \quad (6)$$

The more jagged interpolation function shown on the right of Figure 46 is the attractor of the iterated function system

$$g_1\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}, \qquad g_2\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \quad (7)$$

Although neither of the functions shown in Figure 46 are differentiable at every point of the interval $[0, 1]$, iterated function systems can yield attractors whose graphs are everywhere differentiable functions. For example, the parabolic curve shown on the right of Figure 45 is the attractor of the IFS

$$h_1\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{4} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}, \qquad h_2\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{4} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \quad (8)$$

Notice that the IFS's represented in systems (6), (7), and (8) differ only in the lower right entries of the $2 \times 2$ matrices. It turns out that as long as all these entries (called

vertical scaling factors) are less than one in magnitude, there is a metric on $\Re^2$ with respect to which the mappings $f_1, f_2, g_1, g_2, h_1$, and $h_2$ are contraction mappings (2:214). Hence each of the systems (6), (7), and (8) truly are IFS's, so for each there really is an attractor in $\Re^2$. The contraction mappings exemplified in systems (6), (7), and (8) are called *shear transformations*; because of the upper right 0's in all the $2 \times 2$ matrices, these mappings take lines parallel to the $y$-axis to lines which are also parallel to the $y$-axis (2:214). Consider the attractor $A$ depicted in Figure 46. Given *any* nonempty compact subset of $\Re^2$, infinite iteration of it by the equations of system (7) (as described in Appendix A) yields the nonempty compact attractor $A$. Since $A$ is an attractor, $A = g_1(A) \cup g_2(A)$. In this example, the first half of $A$ is a contracted counterclockwise rotation of all of $A$, and the second half of $A$ is obtained by flipping $A$ vertically, rotating it clockwise, contracting it, and translating it to the second half of the unit interval.

More generally, any function $F$ defined on an interval $[x_0, x_N]$ with known values $F_0, F_1, \ldots, F_N$ at the points $x_0 < x_1 < \ldots < x_N$ can be interpolated by the attractor of an IFS consisting of shear transformations; the graph of such an attractor is called a *fractal interpolation function* (2:220). It is often possible to obtain a close approximation of the underlying function using far fewer than $N$ shear transformations, and the savings in representational data required can be considerable. Consider how succinct is the representation (6) of the function $A$ depicted on the right of Figure 46 compared, for example, to its representation as a sum of sines and cosines. Even if $A$ had been sampled at thousands of points, no closer fractal approximation could be obtained than by ignoring all but the first, middle, and last values. This is because of an inherent self-similarity within $A$ – portions of $A$ are merely affine transformations (albeit sometimes hard to see) of all of $A$.

As will be shown, strange attractors can variously be considered real-valued functions on a subset of a vector space, or vector valued functions on a real interval. It was felt that their chaotic natures might allow them to be represented with fewer shear transformations than embedded data from a random sequence time series. Experimentation, however, provided no strong evidence of such a relationship.

A strange attractor, such as the Lorenz or Henon attractor, represents the steady state, nonperiodic solution of a differential (or difference) equation and as such never intersects itself (18:168). Suppose an attractor $B$ is the solution of an equation in $n$ variables, so that its "native" space is $\Re^n$. The nonintersecting characteristic of $B$ implies that if points on it are represented in $\Re^{n+1}$ by $(n+1)$-tuples consisting of the points and the times at which they are generated, the resulting set of points define a real-valued function on $\Re^n$. Consider for example the two-dimensional Henon attractor depicted in Figure 7. It is generated by successive application of the equations

$$\begin{aligned} x(i+1) &= 1 + y(i) - 1.4x^2(i) \\ y(i+1) &= 0.3x(i) \end{aligned}$$

from some given initial condition (any initial condition will result in the same general form (31:73)). Suppose that at each step in the generation of the Henon attractor, a time is impressed, so that instead of ordered pairs being generated, ordered triples are generated, according to the rules

$$\begin{aligned} t(i+1) &= i+1 \\ x(i+1) &= 1 + y(i) - 1.4x^2(i) \\ y(i+1) &= 0.3x(i) \end{aligned}$$

The resulting set of points may be considered a real-valued function of the pairs $(x_{i+1}, y_{i+1})$, or a vector-valued function of time. The first 25 of them are depicted in Figure 47. If the points were connected sequentially, the resulting graph would be a curve in $\Re^3$.

It is possible to approximate this curve as an attractor of an IFS in much the same way that the parabolic curve of Figure 45 can be approximated by the attractors depicted in

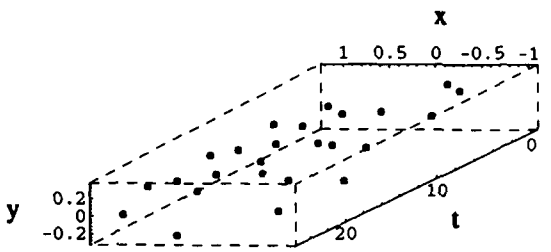



Figure 47. Time-Stamped Henon Attractor

Figure 46 (22). In general, suppose a set of $N + 1$ points

$$\begin{pmatrix} t_0 \\ x_{0,1} \\ x_{0,2} \\ \vdots \\ x_{0,k-1} \end{pmatrix}, \begin{pmatrix} t_1 \\ x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,k-1} \end{pmatrix}, \cdots, \begin{pmatrix} t_N \\ x_{N,1} \\ x_{N,2} \\ \vdots \\ x_{N,k-1} \end{pmatrix} \tag{9}$$

are available in $\Re^k$, with $t_0 < t_1 < \cdots < t_N$. These might be, for example, the 25 points depicted in Figure 47. For any two time indices $t_{i_1} < t_{i_2}$, it is possible to define a contraction mapping $w_i : \Re^k \to \Re^k$ by

$$w_i \begin{pmatrix} t \\ y_1 \\ y_2 \\ \vdots \\ y_{k-1} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & 0 & \cdots & 0 \\ a_{31} & 0 & a_{33} & 0 & \cdots & 0 \\ \vdots & & & & & \\ a_{k1} & 0 & 0 & 0 & \cdots & a_{kk} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \end{pmatrix}$$

which takes all points with time indices between $t_0$ and $t_N$ to points with time indices between $t_{i_1}$ and $t_{i_2}$. The $2k$ parameters $a_{11}$ through $a_{k1}$ and $b_1$ through $b_k$ may be determined by the

$2k$ linear equations in $2k$ unknowns resulting from the endpoint conditions

$$w_i \begin{pmatrix} t_0 \\ x_{0,1} \\ x_{0,2} \\ \vdots \\ x_{0,k-1} \end{pmatrix} = \begin{pmatrix} t_{i_1} \\ x_{i_1,1} \\ x_{i_1,2} \\ \vdots \\ x_{i_1,k-1} \end{pmatrix} \quad \text{and} \quad w_i \begin{pmatrix} t_N \\ x_{N,1} \\ x_{N,2} \\ \vdots \\ x_{N,k-1} \end{pmatrix} = \begin{pmatrix} t_{i_2} \\ x_{i_2,1} \\ x_{i_2,2} \\ \vdots \\ x_{i_2,k-1} \end{pmatrix}$$

Generalizing some work by Mazel and Hayes (23), it turns out that each of the remaining $k-1$ scaling factors $a_{jj}$, $j = 2, 3, \ldots, k$, can be approximated as the ratio of the maximum $y_{j-1}$-deviation of the curve in the interval $[t_{i_1}, t_{i_2}]$ to the maximum $y_{j-1}$-deviation in the interval $[t_0, t_N]$, provided this ratio is less than one. (This ratio can always be made less than one by taking $t_{i_1}$ close enough to $t_{i_2}$.) Consider for example Figure 48, which shows how the scaling factor $a_{jj}$ can be determined for a shear transformation with endpoint conditions

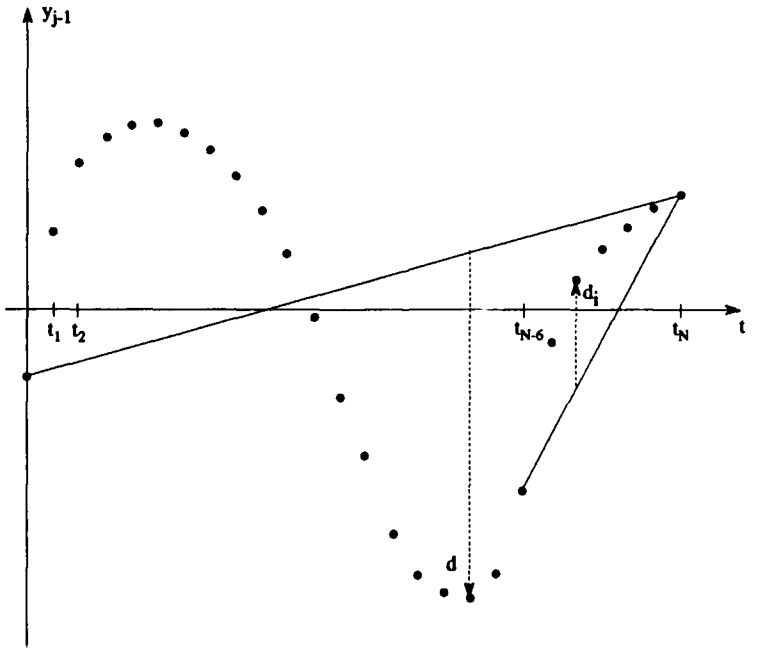



Figure 48. Determining Fractal Interpolation Scaling Factors

at $t_{N-6}$ and $t_N$. First the $N+1$ points given in (9) are projected onto the $(t, y_{j-1})$ plane.

Then the deviation $d$ is determined as the maximum vertical distance from the line joining the endpoints at $t_0$ and $t_N$ to the set of projected points. Then the deviation $d_i$ is determined as the maximum vertical distance from the line joining the endpoints at $t_{N-6}$ and $t_N$. The sign of a deviation is positive if the point of maximizing distance lies above the line joining the endpoints (so that $d_i$ is positive in Figure 48). Conversely, the sign of a deviation is negative if the point of maximizing distance lies below the line joining the endpoints (so that $d$ is negative in Figure 48). The scaling factor $a_{jj} = d_i/d$. Notice that if the interval being considered had been $[t_{N-1}, t_N]$, then there would be zero vertical deviation in this interval, whence $a_{jj} = 0$. Indeed, if the entire interval from $t_0$ to $t_N$ is partitioned point-by-point into subintervals $[t_i, t_{i+1}]$, then for every subinterval, $a_{jj} = 0$ for each $j = 1, 2, \ldots, k-1$. In this case the fractal interpolation of the set of points (9) is piecewise linear in $\Re^k$ and, with $N$ shear transformations, the number $(3k-1) \times N$ of parameters needed to specify the set of points is larger than the number $(N+1) \times k$ of components required to specify the points directly.

A fractal interpolation function is the attractor of a set $\mathcal{S}$ of shear transformations with endpoint conditions which partition the interval $[t_0, t_N]$. It may do either a good or poor job approximating the given set of points (9). A closely approximating fractal interpolation function can usually be fitted recursively to the points (9) using a higher-dimensional version of an approach described by Mazel (23). From the final point at $t_N$, a search is performed backwards through all previous points at times $t_i$, calculating at each $t_i$ the parameters of the shear transformation $w_i$ determined by the interval $[t_i, t_N]$ and recording the maximum separation of the original set of points (9) from their images in $[t_i, t_N]$. The endpoint $t_i \neq t_{N-1}$ which results in the lowest computed error is selected as the new right endpoint (replacing $t_N$) and its corresponding shear transformation $w_i$ becomes one member of the set $\mathcal{S}$. This process is repeated on the left subinterval $[t_0, t_i]$ of $[t_0, t_N]$, isolating the interval $[t_j, t_i]$ and associated shear transformation $w_j$ which results in the least separation of the given set of points from their images under $w_j$ in $[t_j, t_i]$. Eventually the left subinterval is selected in its

entirety, or it becomes $[t_0, t_2]$; in either case, the set $\mathcal{S}$ of shear transformations is completely determined.

Barnsley's collage theorem (2:96) can be used to obtain a bound on the error associated with interpolating the set of points (9) using the set $\mathcal{S}$ of shear transformations. The error bound was not computed in this research. Rather, the principle focus was on comparing the sets $\mathcal{S}$ obtained for pseudo-random time series with the sets $\mathcal{S}$ obtained for chaotic time series, to see if the two types of time series could be distinguished by the gross properties of these sets.

A number of approaches were tried; none were found particularly revealing. In one experiment, the lengths of best end intervals were determined for each of the last 82 points in the embeddings of various time series, each of length 1082. (One of these was a pseudo-random time series generated using Press's routine "ran3" (29:283).) For each embedded time series, the best- fitting end interval of maximum length and the average of the 82 end interval lengths were recorded. In addition to the pseudo-random time series, five chaotic time series were examined: Henon and Lorenz data, and Glass-Mackey data with fractal dimensions of 5.0, 5.5, and 6.0 (15:74). The average end interval lengths ranged from about 11 to about 39, with the pseudo-random data third lowest at about 25.

It might be expected that a random sequence would have a shorter average end interval length than that of chaotic data; after all, there should be no self-similarity within random data to allow compression of the entire data set into large end intervals. Perhaps this result was not observed because the "random" data set used was not truly random but merely pseudo-random.

# *Appendix C. C Program Source Listings*

## *C.1 Description*

The bodies of the programs used are listed in the next section. Subroutines called (many from Press's book *Numerical Recipes in C* (28)) are listed in the last section.

The parameters in the first program, fdfinder4.c, are set at values which seem to reveal well the fractal dimension of the Henon attractor. Applying progressively larger values of ymax, up to about 4000 (or higher, if one is willing to tolerate lengthy computer run times), an apparent convergence to the published value 1.25 (17:193) can be seen. An input file consisting of Henon map $y$-components (accurate to five decimal places) should be used, with the Henon map parameters set to $a = 1.4$, $b = 0.3$ (35:312). The first three values of the $y$-component time series are 0.00000, 0.30000, and $-0.12000$.

The remaining five programs all have parameters set for processing the sleep apnea data set described in Section 3.7. The program casdagli7.c generates all DVS data; it also implements DVS prediction when the parameter kbest is set to a predetermined optimal number of nearest neighbors. The program casdagli13.c implements pruned outliers prediction. Programs casdagli17.c, casdagli18.c, and casdagli19.c are applied sequentially to implement the overlap prediction algorithm; casdagli17.c must be run twice (using different output file names with each run) to produce lists of candidate interval endpoints. The output file goodngbrs produced by program casdagli18.c contains the number of retained intervals as its first entry. This value must be removed from goodngbrs and provided to program casdagli19.c prior to execution of casdagli19.c, which produces the actual predictions in the file casdata. The user may find it convenient to rename the output files with names corresponding to the program which produce them; for example, replace casdata in file casdagli7.c with a file named cas7out, replace casdata in file casdagli13.c with cas13out, etc.

## *C.2 Main Programs*

### *C.2.1 Grassberger and Procaccia Algorithm.*

```
/* This program, fdfinder4.c, finds the fractal dimension of a time series xi
   whose values are in the file cantvals. It implements the algorithm
   described by Grassberger and Procaccia in their article "Measuring the
   Strangeness of Strange Attractors." This program was written by Jim
   Stright in December 1992 and is an adaptation of the Ada program fractall
   which he wrote in 1988. Many of the subroutines are taken from Press et
   al, "Numerical Recipes in C." See closing comments for output format.*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float *vector();
int *ivector();
float **matrix();
void free_vector(),free_ivector(),free_matrix();

void main(void)
{
    FILE *fp;           /* input time series is in file cantvals */
    int D = 5;          /* nbr of elts in each xi vector */
    int ymax = 4000;    /* an upper bound for the nbr of D-tuples */
    float distance;
    float L0 = 1.0;
    int pts_on_line = 8;  /* nbr of pts on "line" whose slope is frac dim */
    float slope;
    int sum;
    float sum1;
    int y3=1;           /* counts nbr of D-tuples */
    int z1=1;           /* counts position within each D-tuple */
    int z3=0;           /* counts nbr of time series vals in input file */
    int i,j,q,r;        /* misc. counters */

    float *L;
    int *CL;
    float **xi;

    L = vector(1,pts_on_line);
    CL = ivector(1,pts_on_line);
```

```
xi = matrix(1,ymax,1,D);

/* open cantvals for input */
if ((fp = fopen("cantvals","r")) == NULL) {
    printf("Cannot open file\n");
    exit(1);
}

/* read in the first D-tuple of data */
while (z1 <= D) {
    fscanf(fp, "%f", &xi[y3][z1]);
    z1 = z1 + 1;
    z3 = z3 + 1;
}
z1 = 1;
y3 = y3 + 1;

/* read in all subsequent data points */
while ((!feof(fp)) && (y3 <= ymax)) {
    while (z1 < D) {
        xi[y3][z1] = xi[y3 - 1][z1 + 1];
        z1 = z1 + 1;
    }
    fscanf(fp, "%f", &xi[y3][D]);
    z3 = z3 + 1;
    z1 = 1;
    y3 = y3 + 1;
}
y3 = y3 - 1;

/* define the small radii L[i] */
for (i = 0; i <= pts_on_line - 1; i++) {
    L[i + 1] = exp(0.5*i - 4.5);    /* L[i + 1] = exp(0.125*i - 2.2); */
}

/* find the CL[i] values */
for (i = 1; i <= pts_on_line; i++) {
    sum = 0;
    for (j = 1; j < y3; j++) {
        for (q = j + 1; q <= y3; q++) {
            sum1 = 0.0;
            for (r = 1; r <= D; r++) {
```

```
                sum1 = sum1 + pow(xi[j][r] - xi[q][r],2.0);
            }
            /* sum1 is now squared distance from xi[j] to xi[q] */
            distance = sqrt(sum1);
            if (distance < L[i])
                sum = sum + 1;
        }
        /* sum is now the nbr of D-tuples near xi[j] */
    }
    /* sum is now the nbr of D-tuples within L[i] of each other */
    CL[i] = sum;
}

/* Output the results */
printf("The number of time series values in cantvals is %d .\n\n", z3);
for (i = 1; i <= pts_on_line; i++) {
    printf("%3.3f", log(L[i]/L0));
    printf("    ");
    printf("%3.3f\n", log(CL[i]));
}
printf("\n");
if (CL[1] == 0.0)
    printf("CL[1] = 0; make L[1] larger\n");
else {
/* output the slopes consecutively, computed from the first point */
    for (i = 2; i <= pts_on_line; i++) {
        slope = (log(CL[i])-log(CL[1])) / (log(L[i]/L0)-log(L[1]/L0));
        printf("%3.3f\n", slope);
    }
    /* last slope output is best "quick" estimate of fractal dimension */
}

free_vector(L,1,pts_on_line);
free_ivector(CL,1,pts_on_line);
free_matrix(xi,1,ymax,1,D);
fclose(fp);
}

/* The program's output, should all go well, will look something like this:

    The number of time series values in cantvals is 10500.
```

```
0.12    -7.53
0.72    -6.14
   .
   .                ( these are the coordinates (ln(L[i]/L0,lnCL[i]) )
   .
1.93    -2.97

2.32    (slope from point 1 to point 2)
2.37    (slope from point 1 to point 3)
  .
  .
  .
2.40    (slope from point 1 to point pts_on_line) */
```

### *C.2.2 DVS Prediction.*

```
/*  This program, casdagli7.c, implements the forecasting algorithm described
    on page 307 of Casdagli's article "Chaos and Deterministic versus
    Stochastic Non-linear Modelling." It was written in May 1993 by Jim
    Stright. Casdagli7.c also provides a prediction of a single value beyond
    the end of the data used for testing (an unknown). It does so using the
    best m & k as found from previous runs of this program. Usually Nt=0 is
    used in the prediction mode, with Nf the number of time series values
    assumed known. Many of the subroutines are taken from Press et al,
    "Numerical Recipes in C." As a predictor, casdagli7.c implements
    "DVS prediction." */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TINY 1.0e-20
double *dvector();
int *ivector();
double **dmatrix();
double moment();
double ludcmp();
double lubksb();
double sort2();
void free_dvector(),free_dmatrix(),free_ivector();

void main(void)
{
```

```
FILE *fp1, *fp2;                /* fp1 is tsdata (input); fp2 is casdata */
int m=14;                       /* embedding dim */
int tau = 1;                    /* delay time */
int T = tau;                    /* forecasting time; not necessarily tau!! */
int kbest = 286;                /* replace with best k, else use 2*(m+1) */
int i,j,ctr1,ctr2,ctr3;         /* counters */
int row,col,l;                  /* more counters */
int k;                          /* nbr of nearest neighbors */
int klast = 0;                  /* This counter is at the nbr (+2*(m+1)) of the
                                   last nearest neighbor incorporated in the
                                   A matrix */
int Nf = 2400;                  /* nbr of time series values in fitting set */
int Nt = 270;                   /* nbr of time series values in testing set */
int Ns = 1;                     /* spacing of the sampled delay vectors */
int n;                          /* required for call to "moment" */
int FLAG = 0;                   /* used in ludcmp for "too large" check */
int kexp = 0;                   /* counter for exponential spacing of k's */
double kbase = 2.0;             /* base for exponential spacing of k's */

double ave,adev,sigma,svar,skew,curt;
        /*  all of these required for call to "moment",
            although only sigma is used in
            casdagli7.c; see Press Ed 2, p.613 */
double *ave_ptr=&ave,*adev_ptr=&adev,*sigma_ptr=&sigma;
double *svar_ptr=&svar,*skew_ptr=&skew,*curt_ptr=&curt;
double *x;

double **A,**Alud,**d,*dhold,*alpha,*b,dnr;
   /* Alud is repeatedly destroyed by ludcmp, dnr is Press's d, p.46 */
int *indx;
int *nbrtested;
double **xhat,**e,*Em,errsum;
x = dvector(1,Nf+Nt);
indx = ivector(1,m+1);
nbrtested = ivector(0,Nf-T-(m-1)*tau-2*(m+1));
A = dmatrix(1,m+1,1,m+1);
Alud = dmatrix(1,m+1,1,m+1);
d = dmatrix(Nf,Nf+Nt,1,Nf-T-(m-1)*tau);
/*  d[i][1] is the distance from vector x[i] to vector x[1+(m-1)*tau];
    d[i][2] is the distance from vector x[i] to vector x[1+(m-1)*tau+1];
        ...
    d[i][Nf-T-(m-1)*tau] is distance from vector x[i] to vector x[Nf-T],
```

```
    before a swap for nearness is performed. */
dhold = dvector(1,Nf-T-(m-1)*tau);
alpha = dvector(1,m+1);
b = dvector(1,m+1);

xhat = dmatrix(0,Nf-T-(m-1)*tau-2*(m+1),Nf+T,Nf+Nt+T);
e = dmatrix(0,Nf-T-(m-1)*tau-2*(m+1),Nf,Nf+Nt);
Em = dvector(0,Nf-T-(m-1)*tau-2*(m+1));

/* open tsdata for input */
if ((fp1 = fopen("tsdata","r")) == NULL) {
    printf("Cannot open file tsdata\n");
    exit(1);
}

/* open casdata for output */
if ((fp2 = fopen("casdata", "w")) == NULL) {
    printf("Cannot open file casdata\n");
    exit(1);
}

/* read in the time series data */
for (ctr1=1; ctr1<=Nf+Nt; ctr1++) {
    fscanf(fp1, "%lf", &x[ctr1]);
}

/* compute distances d[i][j] and load d matrix with nearness indices */
for (i=Nf; i<=Nf+Nt; i++) {
    for (j=1; j<=Nf-T-(m-1)*tau; j++) { /* see indexing note below */
        d[i][j] = fabs(x[i] - x[j+(m-1)*tau]);
        for (ctr1=tau; ctr1<=(m-1)*tau; ctr1=ctr1+tau) {
            if (fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]) > d[i][j]) {
                d[i][j] = fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]);
            }
        }
        /* dist d[i][j] between vctrs x[i] & x[j+(m-1)*tau] is fixed */
    }
    /* the distances d[i][j] are now established for all j */

    /* initialize the index-swap vector dhold */
    for (ctr2=1; ctr2<=Nf-T-(m-1)*tau; ctr2++) {
        dhold[ctr2] = ctr2 + (m-1)*tau;
```

```
        /* now the contents of dhold[1], eg, is 1+(m-1)*tau */
    }

    /* Sort the contents of the vector d[i] and simultaneously sort the
        vector dhold into ascending order of nearness of vectors to x[i];
        see Press Ed 2, page 334. */
    sort2(Nf-T-(m-1)*tau, d[i], dhold);

    /* replace contents of vector d[i] with indices of vectors arranged
        in ascending order of nearness to x[i] */
    for (ctr3=1; ctr3<=Nf-T-(m-1)*tau; ctr3++) {
        d[i][ctr3] = dhold[ctr3];
    }
    /* Now the contents of vector d[i] is the set of indices of vectors
        compared for nearness to vector x[i], arranged in ascending order
        of nearness to x[i]. eg, d[i][1] is the index of the vctr
        nearest x[i]. */
}

/* Find standard dev sigma for the time series; see Press 2, p.613.*/
n = Nf+Nt;
moment(x,n,ave_ptr,adev_ptr,sigma_ptr,svar_ptr,skew_ptr,curt_ptr);

fprintf(fp2,"Data output from program casdagli7.c\n");
fprintf(fp2,"m=%2d\n",m);
fprintf(fp2,"Nf = %d\n",Nf);
fprintf(fp2,"Nt = %d\n",Nt);
fprintf(fp2,"T = tau = %d\n",T);
fprintf(fp2,"average data value ave = %2.5f\n",ave);
fprintf(fp2,"data standard deviation sigma = %2.5f\n",sigma);

/* Initialize the max nbr of vectors to be compared for nearness */
k = 0;
kexp = 0;
while (k <= Nf-T-(m-1)*tau-2*(m+1)) {
    nbrtested[k] = (Nt-T+1)/Ns; /* recall int division trucates */
    k = (int) pow(kbase,kexp);
    kexp = kexp + 1;
}

/* Establish the error matrix e[k][i] */
for (i=Nf; i<=Nf+Nt; i++) {
```

```
/* Initialize the A matrix at k=2*(m+1) */
/* First the diagonal entries: */
A[1][1] = 2*(m+1);
Alud[1][1] = A[1][1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
    A[ctr1][ctr1] = 0.0;
    for (ctr2=1; ctr2<=2*(m+1); ctr2++) {
        A[ctr1][ctr1] = A[ctr1][ctr1]
        + x[(int)(d[i][ctr2])-(ctr1-2)*tau]
        * x[(int)(d[i][ctr2])-(ctr1-2)*tau];
    }
    Alud[ctr1][ctr1] = A[ctr1][ctr1];
}

/* Now the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
    A[1][col] = 0.0;
    for (l=1; l<=2*(m+1); l++) {
        A[1][col] = A[1][col] + x[(int)(d[i][l])-(col-2)*tau];
    }
    Alud[1][col] = A[1][col];
    A[col][1] = A[1][col];
    Alud[col][1] = A[col][1];
}

/* Now initialize the off-diag, off-first-row-or-col entries: */
for (row=2; row<=m; row++) {
    for (col=row+1; col<=(m+1); col++) {
        A[row][col] = 0.0;
        for (l=1; l<=2*(m+1); l++) {
            A[row][col] = A[row][col]
            + x[(int)(d[i][l])-(row-2)*tau]
            * x[(int)(d[i][l])-(col-2)*tau];
        }
        Alud[row][col] = A[row][col];
        A[col][row] = A[row][col];
        Alud[col][row] = A[col][row];
    }
}

/* And last, initialize the b vector, and its equal alpha vector;
    alpha gets replaced with the proper solution when one solves
```

```
    A*alpha = b as A*x=b=alpha; see Press, page 44. */
b[1] = 0.0;
for (l=1; l<=2*(m+1); l=l+1) {
    b[1] = b[1] + x[(int)(d[i][l])+T];
}
alpha[1] = b[1];
for (row=2; row<=(m+1); row++) {
    b[row] = 0.0;
    for (l=1; l<=2*(m+1); l++) {
        b[row] = b[row]
                + x[(int)(d[i][l])-(row-2)*tau]
                * x[(int)(d[i][l])+T];
    }
    alpha[row] = b[row];
}

/* Go after the e[k][i]; may easily change the k indexing to sample
    k's at exponentially spaced intervals. In what follows, k
    equals the nbr of neighbors nearest x[i] minus 2*(m+1). */

klast = 0;
k=0;
kexp = 0;
while (k <= Nf-T-(m-1)*tau-2*(m+1)) {

    /* Nf-T-(m-1)*tau is the nbr of nghbrs in fitting set whose
        "predicted" value is <= Nf */

    /* Update the A matrix */

    /* First update the diagonal entries */
    A[1][1] = 2*(m+1) + k;
    Alud[1][1] = A[1][1];
    for (ctr1=2; ctr1<=(m+1); ctr1++) {
        for (ctr2=1; ctr2<=(k-klast); ctr2++) {
            A[ctr1][ctr1] = A[ctr1][ctr1]
            + x[(int)(d[i][2*(m+1)+klast+ctr2])-(ctr1-2)*tau]
            * x[(int)(d[i][2*(m+1)+klast+ctr2])-(ctr1-2)*tau];
        }
        Alud[ctr1][ctr1] = A[ctr1][ctr1];
    }
```

```
/* Now update the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
    for (l=1; l<=(k-klast); l++) {
        A[1][col] = A[1][col]
            + x[(int)(d[i][2*(m+1)+klast+l])-(col-2)*tau];
    }
    Alud[1][col] = A[1][col];
    A[col][1] = A[1][col];
    Alud[col][1] = A[col][1];
}

/* Now update the off-diag, off-first-row-or-col entries: */
for (row=2; row<=m; row++) {
    for (col=row+1; col<=(m+1); col++) {
        for (l=1; l<=(k-klast); l++) {
            A[row][col] = A[row][col]
            + x[(int)(d[i][2*(m+1)+klast+l])-(row-2)*tau]
            * x[(int)(d[i][2*(m+1)+klast+l])-(col-2)*tau];
        }
        Alud[row][col] = A[row][col];
        A[col][row] = A[row][col];
        Alud[col][row] = A[col][row];
    }
}

/* Finally, update the b vector: */
for (l=1; l<=(k-klast); l++) {
    b[1] = b[1] + x[(int)(d[i][2*(m+1)+klast+l])+T];
}
alpha[1] = b[1];
for (row=2; row<=(m+1); row++) {
    for (l=1; l<=(k-klast); l++) {
        b[row] = b[row]
            + x[(int)(d[i][2*(m+1)+klast+l])-(row-2)*tau]
            * x[(int)(d[i][2*(m+1)+klast+l])+T];
    }
    alpha[row] = b[row];
}

klast = k;

/* Solve the normal eqtns for alpha[1] thru alpha[m+1] */
```

```
        ludcmp(Alud,FLAG,m+1,indx,&dnr);

        if (FLAG==1) {
            alpha[1] = 1000001;
            FLAG=0;}
        else {
            lubksb(Alud,m+1,indx,alpha);
        }

        /* alpha[1],alpha[2],... are now optimum in Casdagli's eqtn 5, if
            the normal equations admit a solution. Otherwise set alpha[1]
            = x[i+T], alpha[2]=alpha[3]=...=alpha[m+1]=0, so that
            xhat[k][i+T] = x[i+T], the exact data value. Also, decrement
            nbrtested so this unusual event isn't included in Em[k]. */
        for (ctr1=1; ctr1<=(m+1); ctr1++) {
            if ((fabs(alpha[ctr1]) > 1000000)||
                (alpha[ctr1] == HUGE_VAL)) {
                nbrtested[k] = nbrtested[k]-1;
                fprintf(fp2,"Error at 1\n");
                alpha[1] = x[i+T];
                for (ctr2=2; ctr2<=(m+1);ctr2++) {
                    alpha[ctr2] = 0.0;
                }
                break;
            }
        }

        xhat[k][i+T] = alpha[1];
        for (ctr1=2; ctr1<=(m+1); ctr1++) {
            xhat[k][i+T] = xhat[k][i+T] + alpha[ctr1]*x[i-(ctr1-2)*tau];
        }
        /* xhat[k][i+T] has now been established */

        if (i <= Nf+Nt-T)
            e[k][i] = fabs(xhat[k][i+T] - x[i+T]);
        else
            e[k][i] = 0.0;

        k = (int) pow(kbase,kexp);
        kexp = kexp + 1;
    } /* closes the loop over k's */
} /* closes the loop over i's */
```

```
    k = 0;
    kexp = 0;
    while (k <= Nf-T-(m-1)*tau-2*(m+1)) {
        errsum = 0.0;
        for (i=Nf; i<=Nf+Nt-T; i++) {
            errsum = errsum + e[k][i]*e[k][i];
        }
        Em[k] = (sqrt(errsum/nbrtested[k]))/sigma;

        fprintf(fp2, "%d", 2*(m+1)+k); /* Output nbr of nearest nghbrs */
        fprintf(fp2,"    ");
        fprintf(fp2, "%1.6f\n", Em[k]); /* Output forecasting error */

        k = (int) pow(kbase,kexp);
        kexp = kexp + 1;
    }

    fprintf(fp2, "Predicted value at time %d is %f\n",
                    Nf+Nt+T,xhat[kbest-2*(m+1)][Nf+Nt+T]);

    free_dvector(x,1,Nf+Nt);
    free_dvector(dhold,1,Nf-T-(m-1)*tau);
    free_dvector(alpha,1,m+1);
    free_dvector(b,1,m+1);
    free_dvector(Em,0,Nf-T-(m-1)*tau-2*(m+1));
    free_ivector(indx,1,m+1);
    free_ivector(nbrtested,0,Nf-T-(m-1)*tau-2*(m+1));
    free_dmatrix(A,1,m+1,1,m+1);
    free_dmatrix(Alud,1,m+1,1,m+1);
    free_dmatrix(d,Nf,Nf+Nt-T,1,Nf-T-(m-1)*tau);
    free_dmatrix(xhat,0,Nf-T-(m-1)*tau-2*(m+1),Nf+T,Nf+Nt);
    free_dmatrix(e,0,Nf-T-(m-1)*tau-2*(m+1),Nf,Nf+Nt-T);
    fclose(fp1);
    fclose(fp2);
}
```

### *C.2.3 Pruned Outliers Prediction.*

```
/* This program, casdagli13.c, adapts the forecasting algorithm described
   on p. 307 of Casdagli's article "Chaos and Deterministic versus Stochastic
   Non-linear Modelling." Written Nov 1993 by Jim Stright, it is a
```

```
    modification of program casdagli7.c. Casdagli13.c provides a prediction of
    a single value beyond the end of the data used for testing (an unknown). It
    does so using the best k (kbest) as found from a previous run of program
    casdagli7.c. Rather than use all kbest nearest nghbrs, it uses only those
    p of them which are close to the hyperplane linear regression best fit of
    the kbest. Fix Nt = 0 in this program; there are Nf known time series
    values. Many of the subroutines are taken from the book by Press et al,
    "Numerical Recipes in C." Casdagli13 c implements "pruned outliers
    prediction." */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TINY 1.0e-20
double *dvector();
int *ivector();
double **dmatrix();
double moment();
double ludcmp();
double lubksb();
double sort2();
void free_dvector(),free_dmatrix(),free_ivector();

void main(void)
{
    FILE *fp1, *fp2;            /* fp1 is tsdata (input); fp2 is casdata */
    int m=14;                   /* embedding dim */
    int tau=1;                  /* delay time */
    int i,j,ctr1,ctr2,ctr3;     /* counters */
    int row,col,l,q;            /* more counters */
    int p;                      /* nbr of nearest nghbrs close to l.reg. line */
    int k;                      /* ctr of nbr of nearest neighbors */
    int kbest = 286;            /* from casdagli7.c's output */
    int klast = 0;              /* This counter is at the nbr (+2*(m+1)) of the
                                   last nearest neighbor incorporated in the
                                   A matrix */
    int Nf = 2670;              /* nbr of time series values in fitting set */
    int Nt = 0;                 /* nbr of time series values in testing set */
    int Ns = 1;                 /* spacing of the sampled delay vectors */
    int T = tau;                /* forecasting time; not necces. tau!! */
    int n;                      /* required for call to "moment" */
    int FLAG = 0;               /* used in ludcmp for "too large" check */
```

```
int kexp = 0;                   /* counter for exponential spacing of k's */
double kbase = 2.0;             /* base for exponential spacing of k's */
double sig;                     /* a measure of nearness to lin.regr. line */
double gamma = 2.0;             /* may want to try eg gamma = 3.0 */

double ave,adev,sigma,svar,skew,curt;
        /*  all of these required for call to "moment",
            although only sigma and ave used in
            casdagli13.c; see Press Ed 2, p.613 */
double *ave_ptr=&ave,*adev_ptr=&adev,*sigma_ptr=&sigma;
double *svar_ptr=&svar,*skew_ptr=&skew,*curt_ptr=&curt;
double *x,*xlr,*er;

double **A,**Alud,**d,*dhold,*alpha,*b,dnr;
   /* Alud is repeatedly destroyed by ludcmp, dnr is Press's d, p.46 */
int *indx;
int *nbrtested;
int *dn; /* used to index nearest nghbrs close to lin. reg. line */
double **xhat,**e,*Em,errsum;
x = dvector(1,Nf+Nt+T);
xlr = dvector(1,kbest);
er = dvector(1,kbest);
indx = ivector(1,m+1);
nbrtested = ivector(0,Nf-T-(m-1)*tau-2*(m+1));
dn = ivector(1,kbest);
A = dmatrix(1,m+1,1,m+1);
Alud = dmatrix(1,m+1,1,m+1);
d = dmatrix(Nf,Nf+Nt,1,Nf-T-(m-1)*tau);
/*  d[i][1] is the distance from vector x[i] to vector x[1+(m-1)*tau];
    d[i][2] is the distance from vector x[i] to vector x[1+(m-1)*tau+1];
        ...
    d[i][Nf-T-(m-1)*tau] is distance from vector x[i] to vector x[Nf-T],
    before a swap for nearness is performed. */
dhold = dvector(1,Nf-T-(m-1)*tau);
alpha = dvector(1,m+1);
b = dvector(1,m+1);

xhat = dmatrix(0,Nf-T-(m-1)*tau-2*(m+1),Nf+T,Nf+Nt+T);
e = dmatrix(0,Nf-T-(m-1)*tau-2*(m+1),Nf,Nf+Nt);
Em = dvector(0,Nf-T-(m-1)*tau-2*(m+1));

/* open tsdata for input */
```

```
if ((fp1 = fopen("tsdata","r")) == NULL) {
    printf("Cannot open file tsdata\n");
    exit(1);
}

/* open casdata for output */
if ((fp2 = fopen("casdata", "w")) == NULL) {
    printf("Cannot open file casdata\n");
    exit(1);
}

/* read in the time series data */
for (ctr1=1; ctr1<=Nf+Nt+T; ctr1++) {
    fscanf(fp1, "%lf", &x[ctr1]);
}

/* compute distances d[i][j] and load d matrix with nearness indices */
for (i=Nf; i<=Nf+Nt; i++) {
    for (j=1; j<=Nf-T-(m-1)*tau; j++) { /* see indexing note below */
        d[i][j] = fabs(x[i] - x[j+(m-1)*tau]);
        for (ctr1=tau; ctr1<=(m-1)*tau; ctr1=ctr1+tau) {
            if (fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]) > d[i][j]) {
                d[i][j] = fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]);
            }
        }
        /* dist d[i][j] between vctrs x[i] & x[j+(m-1)*tau] is fixed */
    }
    /* the distances d[i][j] are now established for all j */

    /* initialize the index-swap vector dhold */
    for (ctr2=1; ctr2<=Nf-T-(m-1)*tau; ctr2++) {
        dhold[ctr2] = ctr2 + (m-1)*tau;
        /* now the contents of dhold[1], eg, is 1+(m-1)*tau */
    }

    /* Sort the contents of the vector d[i] and simultaneously sort the
       vector dhold into ascending order of nearness of vectors to x[i];
       see Press Ed 2, page 334. */
    sort2(Nf-T-(m-1)*tau, d[i], dhold);

    /* replace contents of vector d[i] with indices of vectors arranged
       in ascending order of nearness to x[i] */
```

```
    for (ctr3=1; ctr3<=Nf-T-(m-1)*tau; ctr3++) {
        d[i][ctr3] = dhold[ctr3];
    }
    /* Now the contents of vector d[i] is the set of indices of vectors
        compared for nearness to vector x[i], arranged in ascending order
        of nearness to x[i]; eg, d[i][1] is index of vctr nearest x[i]. */
}

/* Find standard dev sigma for the time series; see Press 2, page 613.*/
n = Nf+Nt;
moment(x,n,ave_ptr,adev_ptr,sigma_ptr,svar_ptr,skew_ptr,curt_ptr);

fprintf(fp2,"Data output from program casdagli13.c\n");
fprintf(fp2,"m=%2d\n",m);
fprintf(fp2,"Nf = %d\n",Nf);
fprintf(fp2,"Nt = %d\n",Nt);
fprintf(fp2,"T = tau = %d\n",T);
fprintf(fp2,"gamma = %f\n",gamma);
fprintf(fp2,"Global optim nbr nearest nghbrs kbest=%3d\n",kbest);
fprintf(fp2,"average data value ave = %2.5f\n",ave);
fprintf(fp2,"data standard deviation sigma = %2.5f\n",sigma);

i = Nf+Nt; /* i remains fixed now at the last known ts value */

/* Initialize the A matrix at k=kbest */
/* First the diagonal entries: */
A[1][1] = kbest;
Alud[1][1] = A[1][1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
    A[ctr1][ctr1] = 0.0;
    for (ctr2=1; ctr2<=kbest; ctr2++) {
        A[ctr1][ctr1] = A[ctr1][ctr1]
        + x[(int)(d[i][ctr2])-(ctr1-2)*tau]
        * x[(int)(d[i][ctr2])-(ctr1-2)*tau];
    }
    Alud[ctr1][ctr1] = A[ctr1][ctr1];
}

/* Now the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
    A[1][col] = 0.0;
    for (l=1; l<=kbest; l++) {
```

```
            A[1][col] = A[1][col] + x[(int)(d[i][l])-(col-2)*tau];
        }
        Alud[1][col] = A[1][col];
        A[col][1] = A[1][col];
        Alud[col][1] = A[col][1];
    }

    /* Now initialize the off-diag, off-first-row-or-col entries: */
    for (row=2; row<=m; row++) {
        for (col=row+1; col<=(m+1); col++) {
            A[row][col] = 0.0;
            for (l=1; l<=kbest; l++) {
                A[row][col] = A[row][col]
                + x[(int)(d[i][l])-(row-2)*tau]
                * x[(int)(d[i][l])-(col-2)*tau];
            }
            Alud[row][col] = A[row][col];
            A[col][row] = A[row][col];
            Alud[col][row] = A[col][row];
        }
    }

    /* And last, initialize the b vector, and its equal alpha vector;
        alpha gets replaced with the proper solution when A*alpha = b is
        solved as A*x=b=alpha; see Press, page 44. */
    b[1] = 0.0;
    for (l=1; l<=kbest; l++) {
        b[1] = b[1] + x[(int)(d[i][l])+T];
    }
    alpha[1] = b[1];
    for (row=2; row<=(m+1); row++) {
        b[row] = 0.0;
        for (l=1; l<=kbest; l++) {
            b[row] = b[row]
                    + x[(int)(d[i][l])-(row-2)*tau]
                    * x[(int)(d[i][l])+T];
        }
        alpha[row] = b[row];
    }
    k = kbest - 2*(m+1);

    /* Solve the normal eqtns for alpha[1] thru alpha[m+1] */
```

```
ludcmp(Alud,FLAG,m+1,indx,&dnr);

if (FLAG==1) {
    alpha[1] = 1000001;
    FLAG=0;}
else {
    lubksb(Alud,m+1,indx,alpha);
}

/* alpha[1],alpha[2],... are now optimum in Casdagli's eqtn 5, if
    the normal equations admit a solution. Otherwise set alpha[1]
    = x[i+T], alpha[2]=alpha[3]=...=alpha[m+1]=0, so that
    xhat[k][i+T] = x[i+T], the exact data value. Also, decrement
    nbrtested so this unusual event isn't included in Em[k]. */

for (ctr1=1; ctr1<=(m+1); ctr1++) {
    if ((fabs(alpha[ctr1]) > 1000000)||
        (alpha[ctr1] == HUGE_VAL)) {
        nbrtested[k] = nbrtested[k]-1;
        alpha[1] = x[i+T];
        for (ctr2=2; ctr2<=(m+1);ctr2++) {
            alpha[ctr2] = 0.0;
        }
        break;
    }
}

for (q=1; q <= kbest; q++) {
    xlr[q] = alpha[1];
    for (ctr1=2; ctr1 <= (m+1); ctr1++) {
        xlr[q] = xlr[q] + alpha[ctr1] *
                x[(int)(d[i][q]) - (ctr1-2)*tau];
    }
}
/* The value of each of the nearest nghbrs x[(int)(d[i][q])] of
    x[i] on the linear regression hyperplane is now established
    as xlr[q]. */

for (q=1; q <= kbest; q++) {
    er[q] = fabs(xlr[q] - x[(int)(d[i][q])+T]);
}
```

```
/* The error er[q] between the next value of the time series
    corresponding to nghbr q and the value of its linear
    regression is now established for all q. */

moment(er,kbest,ave_ptr,adev_ptr,sigma_ptr,svar_ptr,skew_ptr,
        curt_ptr);
/* Now sigma is the standard deviation of the er[q], and ave
    is the average value of the er[q]. */

sig = gamma*sigma; /* may want to try, eg, sig=2.0*sigma */

p=1;
q=1;
while (q <= kbest) {
    if (er[q] < ave + sig) {
        dn[p] = (int)(d[i][q]);
        p = p+1;
        }
    q = q+1;
}
p = p - 1;
/* Now the p nearest nghbrs to vctr x[Nf+Nt] (p<=kbest) with next
    vals within ave+sig of the regression hyperplane determined by
    the kbest nearest nghbrs to vctr x[Nf+Nt] are the vectors
    x[dn[1]],x[dn[2]],...,x[dn[p]] (given in descending order
    of nearness; eg, x[dn[1]] is nearest x[Nf+Nt]). */

if (p < 2*(m+1)) {
    fprintf(fp2,"Too few near with p=%2d; INCREASE GAMMA!\n",p);
    }
else {
    /* Reestablish the A matrix at k=p. */
    /* First the diagonal entries: */
    A[1][1] = p;
    Alud[1][1] = A[1][1];
    for (ctr1=2; ctr1<=(m+1); ctr1++) {
        A[ctr1][ctr1] = 0.0;
        for (ctr2=1; ctr2<=p; ctr2++) {
            A[ctr1][ctr1] = A[ctr1][ctr1]
            + x[dn[ctr2]-(ctr1-2)*tau]
            * x[dn[ctr2]-(ctr1-2)*tau];
        }
```

```
    Alud[ctr1][ctr1] = A[ctr1][ctr1];
}

/* Now the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
    A[1][col] = 0.0;
    for (l=1; l<=p; l++) {
        A[1][col] = A[1][col] + x[dn[l]-(col-2)*tau];
    }
    Alud[1][col] = A[1][col];
    A[col][1] = A[1][col];
    Alud[col][1] = A[col][1];
}

/* Now init. the off-diag, off-first-row-or-col entries: */
for (row=2; row<=m; row++) {
    for (col=row+1; col<=(m+1); col++) {
        A[row][col] = 0.0;
        for (l=1; l<=p; l++) {
            A[row][col] = A[row][col]
            + x[dn[l]-(row-2)*tau]
            * x[dn[l]-(col-2)*tau];
        }
        Alud[row][col] = A[row][col];
        A[col][row] = A[row][col];
        Alud[col][row] = A[col][row];
    }
}

/* And last, init. the b vctr, and its equal alpha vector;
    alpha gets replaced with the right sol. when one solves
    A*alpha = b as A*x=b=alpha; see Press, page 44. */
b[1] = 0.0;
for (l=1; l<=p; l++) {
    b[1] = b[1] + x[dn[l]+T];
}
alpha[1] = b[1];
for (row=2; row<=(m+1); row++) {
    b[row] = 0.0;
    for (l=1; l<=p; l++) {
        b[row] = b[row]
                + x[dn[l]-(row-2)*tau]
```

```
                    * x[dn[l]+T];
            }
            alpha[row] = b[row];
        }
    }

/* Solve the normal eqtns for alpha[1] thru alpha[m+1] */

ludcmp(Alud,FLAG,m+1,indx,&dnr);

if (FLAG==1) {
    alpha[1] = 1000001;
    FLAG=0;}
else {
    lubksb(Alud,m+1,indx,alpha);
}

/* alpha[1],alpha[2],... are now optimum in Casdagli's eqtn 5, if
    the normal equations admit a solution. Otherwise set alpha[1]
    = x[i+T], alpha[2]=alpha[3]=...=alpha[m+1]=0, so that
    xhat[k][i+T] = x[i+T], the exact data value. Also, decrement
    nbrtested so this unusual event isn't included in Em[k]. */

for (ctr1=1; ctr1<=(m+1); ctr1++) {
    if ((fabs(alpha[ctr1]) > 1000000)||
        (alpha[ctr1] == HUGE_VAL)) {
        nbrtested[k] = nbrtested[k]-1;
        alpha[1] = x[i+T];
        for (ctr2=2; ctr2<=(m+1);ctr2++) {
            alpha[ctr2] = 0.0;
        }
        break;
    }
}

xhat[k][i+T] = alpha[1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
    xhat[k][i+T] = xhat[k][i+T] + alpha[ctr1]*x[i-(ctr1-2)*tau];
}
/* xhat[k][i+T] has now been established; it is the predicted ts
    value at time i+T = Nf+Nt+T. */
```

```
    fprintf(fp2, "lin. regr. hyperplane errors ave = %3.5f\n",ave);
    fprintf(fp2, "nbr nghbrs close to lin. regr. hyperplane = p = %2d\n",p);
    fprintf(fp2, "The predicted value at time %d is %f\n",
                    Nf+Nt+T,xhat[kbest-2*(m+1)][Nf+Nt+T]);
    fprintf(fp2, "The actual value at time %d is %f\n",Nf+Nt+T,x[Nf+Nt+T]);

    free_dvector(x,1,Nf+Nt+T);
    free_dvector(xlr,1,kbest);
    free_dvector(er,1,kbest);
    free_dvector(dhold,1,Nf-T-(m-1)*tau);
    free_dvector(alpha,1,m+1);
    free_dvector(b,1,m+1);
    free_dvector(Em,0,Nf-T-(m-1)*tau-2*(m+1));
    free_ivector(indx,1,m+1);
    free_ivector(nbrtested,0,Nf-T-(m-1)*tau-2*(m+1));
    free_ivector(dn,1,kbest);
    free_dmatrix(A,1,m+1,1,m+1);
    free_dmatrix(Alud,1,m+1,1,m+1);
    free_dmatrix(d,Nf,Nf+Nt-T,1,Nf-T-(m-1)*tau);
    free_dmatrix(xhat,0,Nf-T-(m-1)*tau-2*(m+1),Nf+T,Nf+Nt);
    free_dmatrix(e,0,Nf-T-(m-1)*tau-2*(m+1),Nf,Nf+Nt-T);
    fclose(fp1);
    fclose(fp2);
}
```

### *C.2.4 Overlap Prediction.*

```
/* This program, casdagli17.c, uses the best m and k as found from previous
   runs of casdagli7.c to prepare for pred. of Nf+Nt+1 = Nf+1 for a given
   value of T = tau (a common best compromise m and k are used throughout
   casdagli17.c and casdagli18.c). Written Jan 1994 by Jim Stright, it is
   derived from casdagli7.c. Progam casdagli17 c provides as outputs the
   files tauxdata containing the right endpoints of the k ngbrs nearest
   the point Nf+Nt+1-taux, where taux is the value of tau = T used and "x"
   is replaced with 1 and 2, corresponding to short and long delays tau.
   Many of the subroutines are taken from the book by Press et al,
   "Numerical Recipes in C." Two runs of this program produce the files
   tau1data and tau2data and constitute the first part of implementing
   "overlap predictions." The file tau1data is pruned against tau2data
   using the program "casdagli18.c." Casdagli17.c is the first of three
   programs used to implement "overlap prediction;" after casdagli18.c is
   executed, the actual predicted value is obtained using casdagli19.c. */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
double *dvector();
double **dmatrix();
double sort2();
void free_dvector(),free_dmatrix();

void main(void)
{
    FILE *fp1, *fp2;                /* fp1 is tsdata (input); fp2 is tauxdata */
    /* CHANGE NAME TAUXDATA BELOW WITH EACH CHANGE IN T = TAU */
    int m = 14;                     /* embedding dim; from casdagli7.c's output */
    int k = 158;                    /* from casdagli7.c's output; compromise kbest */
    int Nf = 2670;                  /* nbr of time series values in fitting set */
    int Nt = 0;                     /* nbr of ts vals in testing set; fixed at 0 */
    int T = 4;                      /* T = tau; eg, 1st tau1 = 4, then tau2 = 7 */
    int tau = T;                    /* delay time; must equal T now */
    int i;                          /* same as Nf + Nt + 1 - tau */
    int j,ctr1,ctr2,ctr3;           /* counters */

    double *x;
    double *dhold;
    double **d;

    x = dvector(1,Nf+Nt);
    dhold = dvector(1,Nf+Nt-m*tau);
    d = dmatrix(Nf+Nt+1-tau,Nf+Nt+1-tau,1,Nf+Nt-m*tau);
    /*  d[i][1] is the distance from vector x[i] to vector x[1+(m-1)*tau];
        d[i][2] is the distance from vector x[i] to vector x[1+(m-1)*tau+1];
            ...
        d[i][Nf+Nt-m*tau] is distance from vctr x[i] to vctr x[Nf+Nt-tau],
        before a swap for nearness is performed. */

    /* open tsdata for input */
    if ((fp1 = fopen("tsdata","r")) == NULL) {
        printf("Cannot open file tsdata\n");
        exit(1);
    }

    /* open tauxdata for output */
```

```
if ((fp2 = fopen("tau1data", "w")) == NULL) {
    printf("Cannot open file tauxdata\n");
    exit(1);
}

/* read in the time series data */
for (ctr1=1; ctr1<=Nf+Nt; ctr1++) {
    fscanf(fp1, "%lf", &x[ctr1]);
}

/* compute distances d[i][j] and load d matrix with nearness indices */
i = Nf + Nt + 1 - tau;
for (j=1; j<=Nf-T-(m-1)*tau; j++) { /* see indexing note below */
    d[i][j] = fabs(x[i] - x[j+(m-1)*tau]);
    for (ctr1=tau; ctr1<=(m-1)*tau; ctr1=ctr1+tau) {
        if (fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]) > d[i][j]) {
            d[i][j] = fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]);
        }
    }
    /* dist d[i][j] between vctrs x[i] & x[j+(m-1)*tau] is fixed */
}
/* the distances d[i][j] are now established for all j */

/* initialize the index-swap vector dhold */
for (ctr2=1; ctr2<=Nf-T-(m-1)*tau; ctr2++) {
    dhold[ctr2] = ctr2 + (m-1)*tau;
    /* now the contents of dhold[1], eg, is 1+(m-1)*tau */
}

/* Sort the contents of the vector d[i] and simultaneously sort the
    vector dhold into ascending order of nearness of vectors to x[i];
    see Press Ed 2, page 334. */
sort2(Nf-T-(m-1)*tau, d[i], dhold);

/* replace contents of vector d[i] with indices of vectors arranged
    in ascending order of nearness to x[i] */
for (ctr3=1; ctr3<=Nf-T-(m-1)*tau; ctr3++) {
    d[i][ctr3] = dhold[ctr3];
}
/* Now the contents of vector d[i] is the set of indices of vectors
    compared for nearness to vector x[i], arranged in ascending order
    of nearness to x[i]; eg, d[i][1] is index of vctr nearest x[i]. */
```

```
    for (j=1; j <= k; j++) {
        fprintf(fp2,"%d\n",(int)(d[i][j]));
    }

    free_dvector(x,1,Nf+Nt);
    free_dvector(dhold,1,Nf+Nt-m*tau);
    free_dmatrix(d,Nf+Nt+1-tau,Nf+Nt+1-tau,1,Nf+Nt-m*tau);
    fclose(fp1);
    fclose(fp2);
}


/*  This program, casdagli18.c, inputs two sets of right endpoints of intervals
    corresponding to two sets of nearest ngbrs for two different values of tau,
    here denoted tau1 and tau2. The left endpoints corresponding to the most
    distant past component of each interval are computed and stored. The
    resulting sets of intervals are checked for overlap; the strategy is to
    retain small intervals which overlap bigger ones. The retained right
    endpoints are output to be used for prediction in program casdagli19.c.
    Written Jan 1994 by Jim Stright, program casdagli18.c is used after the
    program casdagli17.c as one of three programs which together implement
    "overlap prediction." It uses compromise values of k & m as found
    from previous runs of casdagli7.c; it also relies on casdagli7.c for the
    best two delays tau1 and tau2 (tau2 > tau1) to use for prediction. Some
    subroutines are taken from Press et al, "Numerical Recipes in C." */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int *ivector();
void free_ivector();

void main(void)
{
    FILE *fp1, *fp2;            /* fp1 is tau1data, fp2 is tau2data (inputs) */
    FILE *fp3;                  /* fp3 is goodngbrs (output) */
    int m = 14;                 /* embedding dim; from casdagli7.c's output */
    int k = 158;                /* from casdagli7.c's output; compromise k */
    int tau1= 4;                /* delay time for small intervals */
    int tau2 = 7;               /* delay time for big intervals */
    int a = 0;                  /* nbr small intervals retained; initially 0 */
```

```
int i = 1;                      /* small interval counter */
int j = 1;                      /* big interval counter */
int q;                          /* general purpose counter */
int *srep;                      /* right endpoints of input small intervals */
int *brep;                      /* right endpoints of input big intervals */
int *slep;                      /* left endpoints of input small intervals */
int *blep;                      /* left endpoints of input big intervals */
int *dn;                        /* right endpoints of output small intervals */

srep = ivector(1,k);
brep = ivector(1,k);
slep = ivector(1,k);
blep = ivector(1,k);
dn = ivector(1,k);

/* open tau1data for input */
if ((fp1 = fopen("tau1data","r")) == NULL) {
    printf("Cannot open file tau1data\n");
    exit(1);
}

/* open tau2data for input */
if ((fp2 = fopen("tau2data","r")) == NULL) {
    printf("Cannot open file tau2data\n");
    exit(1);
}

/* open goodngbrs for output */
if ((fp3 = fopen("goodngbrs", "w")) == NULL) {
    printf("Cannot open file goodngbrs\n");
    exit(1);
}

/* read in the tau1data */
for (q=1; q<=k; q++) {
    fscanf(fp1, "%d", &srep[q]);
}

/* read in the tau2data */
for (q=1; q<=k; q++) {
    fscanf(fp2, "%d", &brep[q]);
}
```

```
/* set the small left endpoints */
for (q=1; q<=k; q++) {
    slep[q] = srep[q] - (m-1)*tau1;
}

/* set the big left endpoints */
for (q=1; q<=k; q++) {
    blep[q] = brep[q] - (m-1)*tau2;
}

while (i <= k) {
    /* If it is possible to reject a small interval and still obtain
       enough small itervals for linear regression, do so if there
       is any overlap. */

    if (a + (k-i+1) > 2*(m+1)) {
        while (j <= k) { /* check intervals i and j for overlap */
            if (srep[i] < blep[j]) {
                j = j+1;
                }
            else {
                if (slep[i] <= brep[j]) {
                    a = a+1;
                    dn[a] = srep[i];
                    break;
                    }
                else j = j + 1;
                }
        }
        j = 1;
        }
    else {
        for (q=i; q <= k; q++) {
            a = a + 1;
            dn[a] = srep[q];
        }
        break;
        }
    i = i + 1;
}
/* Now dn[1], dn[2], ... ,dn[a] are all of the retained right endpts. */
```

```
    fprintf(fp3,"a = %d\n",a);

    for (q=1; q <= a; q++) {
        fprintf(fp3,"%d\n",dn[q]);
    }

    free_ivector(srep,1,k);
    free_ivector(brep,1,k);
    free_ivector(slep,1,k);
    free_ivector(blep,1,k);
    free_ivector(dn,1,k);
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
}


/*  This program, casdagli19.c, modifies the forecasting algorithm described
    on p. 307 of Casdagli's article "Chaos and Deterministic versus Stochastic
    Non-linear Modelling." Written Jan 1994 by Jim Stright, it is also a
    modification of casdagli7.c. Casdagli19.c provides a prediction of
    a single value beyond the end of the data used for testing . It does so
    using the best m and k (kbest) as found from previous runs of program
    casdagli7.c. Rather than use all kbest nearest nghbrs, it uses only those
    knext of them derived from overlapping intervals from the best 2 tau values
    (those used in program casdagli18.c). The smaller T=tau (tau1) is used;
    also used is the number "a" of neighbors which is taken from the file
    "goodngbrs" provided by program casdagli18.c. Casdagli19.c is the final
    one of three programs used to implement "overlap prediction;" the order of
    execution is casdagli17.c (twice), casdagli18.c, then casdagli19.c. Many
    of the subroutines are taken from Press et al, "Numerical Recipes in C."*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TINY 1.0e-20
#define BIG 1.0e20
double *dvector();
int *ivector();
double **dmatrix();
double moment();
```

```
double ludcmp();
double lubksb();
void free_dvector(),free_dmatrix(),free_ivector();

void main(void)
{
    FILE *fp1, *fp2;            /* fp1 is tsdata, fp2 is goodngbrs (inputs) */
    FILE *fp3;                  /* fp3 is casdata (output) */
    int koriginal = 158;        /* original nbr of intervals; from casdagli17.c */
    int a = 151;                /* nbr kept intrvls (STRIPPED FROM GOODNGBRS!) */
    int m=14;                   /* embedding dim; from casdagli7.c's output */
    int Nf = 2670;              /* nbr of time series values in fitting set */
    int Nt = 0;                 /* nbr of ts vals in testing set; fixed at 0 */
    int T = 4;                  /* forecasting time; = tau1 of casdagli18.c */
    int tau = T;                /* delay time; must equal T now */
    int tau1 = T;               /* the smaller of the two tau values */
    int tau2 = 7;               /* larger tau value; can get from casdagli18.c */
    int lastpred = 0;           /* number of desired predictions minus one */
    int i,j,ctr1,ctr2,ctr3;     /* counters */
    int row,col,l,q;            /* more counters */
    int k = a;                  /* same as a above */
    int kbest = a;              /* same as a above */
    int knext = a;              /* same as a above */
    int klast = 0;              /* This counter is at the nbr (+2*(m+1)) of the
                                   last nearest neighbor incorporated in the
                                   A matrix */
    int Ns = 1;                 /* spacing of the sampled delay vectors */
    int n;                      /* required for call to "moment" */
    int FLAG = 0;               /* used in ludcmp for "too large" check */
    int kexp = 0;               /* counter for exponential spacing of k's */
    double kbase = 2.0;         /* base for exponential spacing of k's */
    double xhatn;               /* the (repeatedly replaced) predicted value */
    double ave,adev,sigma,svar,skew,curt;
                                /* all of these required for call to "moment",
                                   although only ave and sigma are used in
                                   casdagli19.c; see Press Ed 2, p.613 */

    double *ave_ptr=&ave,*adev_ptr=&adev,*sigma_ptr=&sigma;
    double *svar_ptr=&svar,*skew_ptr=&skew,*curt_ptr=&curt;
    double *x,*xlr,*er;
    double *dadd;     /* dadd[j] is distance from (m+1)-tuple ending at
                         x[j+m*tau] to the final (m+1)-tuple in the time series */
```

```
double **A,**Alud,**d,*alpha,*b,dnr;
    /* Alud is repeatedly destroyed by ludcmp, dnr is Press's d, p.46 */
double *dhold; /* used to index nearest m-tuples */
double *dahold; /* used to index nearest (m+1)-tuples */
int *indx;
int *nbrtested;
int *dn; /* used to index nearest nghbrs close to lin. reg. line */
int *dnn; /* used to index nghbrs a little farther away */
int *gngbrs; /* the nbr of nghbrs used, and their values */

x = dvector(1,Nf+Nt+(lastpred+1)*T);
xlr = dvector(1,kbest);
er = dvector(1,kbest);
dadd = dvector(1-tau,Nf-T-m*tau);
dhold = dvector(1,Nf-T-(m-1)*tau);
dahold = dvector(1-tau,Nf-T-m*tau);
indx = ivector(1,m+1);
dn = ivector(1,kbest);
dnn = ivector(1,kbest);
gngbrs = ivector(1,a);
A = dmatrix(1,m+1,1,m+1);
Alud = dmatrix(1,m+1,1,m+1);
d = dmatrix(Nf+Nt,Nf+Nt+lastpred*T,1,Nf-T-(m-1)*tau);
/*  d[i][1] is the distance from vector x[i] to vector x[1+(m-1)*tau];
    d[i][2] is the distance from vector x[i] to vector x[1+(m-1)*tau+1];
        ...
    d[i][Nf-T-(m-1)*tau] is distance from vector x[i] to vector x[Nf-T],
    before a swap for nearness is performed. */
alpha = dvector(1,m+1);
b = dvector(1,m+1);

/* open tsdata for input */
if ((fp1 = fopen("tsdata","r")) == NULL) {
    printf("Cannot open file tsdata\n");
    exit(1);
}

/* open goodngbrs for input */
if ((fp2 = fopen("goodngbrs","r")) == NULL) {
    printf("Cannot open file goodngbrs\n");
    exit(1);
}
```

```
/* open casdata for output */
if ((fp3 = fopen("casdata", "w")) == NULL) {
    printf("Cannot open file casdata\n");
    exit(1);
}

/* read in the time series data */
for (ctr1=1; ctr1<=Nf+Nt+1; ctr1++) {
    fscanf(fp1, "%lf", &x[ctr1]);
}

/* read in the right endpoints of neighboring intervals data */
for (ctr1=1; ctr1<=a; ctr1++) {
    fscanf(fp2, "%d", &gngbrs[ctr1]);
}

/* Find standard dev sigma for the time series; see Press, page 613. */
n = Nf+Nt;
moment(x,n,ave_ptr,adev_ptr,sigma_ptr,svar_ptr,skew_ptr,curt_ptr);

fprintf(fp3,"Data output from program casdagli19.c\n");
fprintf(fp3,"m=%2d\n",m);
fprintf(fp3,"using tau1=%d and tau2=%d\n",tau1,tau2);
fprintf(fp3,"original nbr nearest nghbrs = %d\n",koriginal);
fprintf(fp3,"nbr nearest nghbrs retained a = %3d\n",a);
fprintf(fp3,"average data value ave = %2.5f\n",ave);
fprintf(fp3,"data standard deviation sigma = %2.5f\n",sigma);
fprintf(fp3,"ts index    true val    pred val\n");

i=Nf+Nt+1-T; /* A fixed value in this program */

/* Proceed to predict from the m-tuple ending at x[i], using as
    nearest ngbrs the knext ngbrs nearest to x[i] (with components
    spaced tau units apart). That is, use as the m-tuple neighbors
    of the m-tuple x[i], the m-tuples with indices gngbrs[1],
    gngbrs[2], ... , gngbrs[knext]. */

/* Establish the A matrix at k=knext. */
/* First the diagonal entries: */
A[1][1] = knext;
Alud[1][1] = A[1][1];
```

```
for (ctr1=2; ctr1<=(m+1); ctr1++) {
    A[ctr1][ctr1] = 0.0;
    for (ctr2=1; ctr2<=knext; ctr2++) {
        A[ctr1][ctr1] = A[ctr1][ctr1]
        + x[gngbrs[ctr2]-(ctr1-2)*tau]
        * x[gngbrs[ctr2]-(ctr1-2)*tau];
    }
    Alud[ctr1][ctr1] = A[ctr1][ctr1];
}

/* Now the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
    A[1][col] = 0.0;
    for (l=1; l<=knext; l++) {
        A[1][col] = A[1][col] + x[gngbrs[l]-(col-2)*tau];
    }
    Alud[1][col] = A[1][col];
    A[col][1] = A[1][col];
    Alud[col][1] = A[col][1];
}

/* Now initialize the off-diag, off-first-row-or-col entries: */
for (row=2; row<=m; row++) {
    for (col=row+1; col<=(m+1); col++) {
        A[row][col] = 0.0;
        for (l=1; l<=knext; l++) {
            A[row][col] = A[row][col]
            + x[gngbrs[l]-(row-2)*tau]
            * x[gngbrs[l]-(col-2)*tau];
        }
        Alud[row][col] = A[row][col];
        A[col][row] = A[row][col];
        Alud[col][row] = A[col][row];
    }
}

/* And last, initialize the b vector, and its equal alpha vector;
    alpha gets replaced with the right solution when
    A*alpha = b is solved as A*x=b=alpha; see Press, page 44.*/
b[1] = 0.0;
for (l=1; l<=knext; l++) {
    b[1] = b[1] + x[gngbrs[l]+T];
```

```
}
alpha[1] = b[1];
for (row=2; row<=(m+1); row++) {
    b[row] = 0.0;
    for (l=1; l<=knext; l++) {
        b[row] = b[row]
                + x[gngbrs[l]-(row-2)*tau]
                * x[gngbrs[l]+T];
    }
    alpha[row] = b[row];
}

/* Solve the normal eqtns for alpha[1] thru alpha[m+1] */
ludcmp(Alud,FLAG,m+1,indx,&dnr);

if (FLAG==1) {
    alpha[1] = 1000001;
    FLAG=0;}
else {
    lubksb(Alud,m+1,indx,alpha);
}

/* alpha[1],alpha[2],... are now optimum in Casdagli's eqtn 5, if
    the normal equations admit a solution. Otherwise set alpha[1]
    = x[i], alpha[2]=alpha[3]=...=alpha[m+1]=0, so that
    xhatn = x[i], the previous data value. */

for (ctr1=1; ctr1<=(m+1); ctr1++) {
    if ((fabs(alpha[ctr1]) > 1000000)||
        (alpha[ctr1] == HUGE_VAL)) {
        alpha[1] = x[i];
        for (ctr2=2; ctr2<=(m+1);ctr2++) {
            alpha[ctr2] = 0.0;
        }
        break;
    }
}

xhatn = alpha[1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
    xhatn = xhatn + alpha[ctr1]*x[i-(ctr1-2)*tau];
}
```

```
    /* xhatn has now been established; it is the predicted time series
       value at time i+T. */

    fprintf(fp3,"%5d      %f      %f\n",i+T,x[i+T],xhatn);
    free_dvector(x,1,Nf+Nt+(lastpred+1)*T);
    free_dvector(xlr,1,kbest);
    free_dvector(er,1,kbest);
    free_dvector(dadd,1-tau,Nf-T-m*tau);
    free_dvector(alpha,1,m+1);
    free_dvector(b,1,m+1);
    free_dvector(dhold,1,Nf-T-(m-1)*tau);
    free_dvector(dahold,1-tau,Nf-T-m*tau);
    free_ivector(indx,1,m+1);
    free_ivector(dn,1,kbest);
    free_ivector(dnn,1,kbest);
    free_ivector(gngbrs,1,a);
    free_dmatrix(A,1,m+1,1,m+1);
    free_dmatrix(Alud,1,m+1,1,m+1);
    free_dmatrix(d,Nf+Nt,Nf+Nt+lastpred*T,1,Nf-T-(m-1)*tau);
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
}
```

### *C.3 Subroutines*

```
double moment(data,n,ave,adev,sdev,svar,skew,curt)
int n;
double *data,*ave,*adev,*sdev,*svar,*skew,*curt;
{
    int i,j;
    double s,p;
    void nrerror();

    if (n <= 1) nrerror("n must be at least 2 in MOMENT");

    s=0.0;
    for (j=1;j<=n;j++) s += data[j];
    *ave=s/n;
    *adev=(*svar)=(*skew)=(*curt)=0.0;
    for (j=1;j<=n;j++) {
```

```
        *adev += fabs(s=data[j]-(*ave));
        *svar += (p=s*s);
        *skew += (p *= s);
        *curt += (p *= s);
    }
    *adev /= n;
    *svar /= (n-1);
    *sdev=sqrt(*svar);
    if (*svar) {
        *skew /= (n*(*svar)*(*sdev));
        *curt=(*curt)/(n*(*svar)*(*svar))-3.0;
    } else nrerror("No skew/kurtosis when variance = 0 (in MOMENT)");
}

double ludcmp(a,FLAG,n,indx,d)
int FLAG,n,*indx;
double **a,*d;
{
    int i,imax,j,k;
    double big,dum,sum,temp;
    double *vv,*dvector();
    void nrerror(),free_dvector();

    vv=dvector(1,n);
    *d=1.0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if ((temp=fabs(a[i][j])) > big) big=temp;
        if (big == 0.0){
            FLAG = 1;
            return;
        }
        vv[i]=1.0/big;
    }
    for (j=1;j<=n;j++) {
        for (i=1;i<j;i++) {
            sum=a[i][j];
            for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
        }
        big=0.0;
```

```
        for (i=j;i<=n;i++) {
            sum=a[i][j];
            for (k=1;k<j;k++)
                sum -= a[i][k]*a[k][j];
            a[i][j]=sum;
            if ( (dum=vv[i]*fabs(sum)) >= big) {
                big=dum;
                imax=i;
            }
        }
        if (j != imax) {
            for (k=1;k<=n;k++) {
                dum=a[imax][k];
                a[imax][k]=a[j][k];
                a[j][k]=dum;
            }
            *d = -(*d);
            vv[imax]=vv[j];
        }
        indx[j]=imax;
        if (a[j][j] == 0.0) a[j][j]=TINY;
        if (j != n) {
            dum=1.0/(a[j][j]);
            for (i=j+1;i<=n;i++) a[i][j] *= dum;
        }
    }
    free_dvector(vv,1,n);
}

double lubksb(a,n,indx,b)
double **a,b[];
int n,*indx;
{
    int i,ii=0,ip,j;
    double sum;

    for (i=1;i<=n;i++) {
        ip=indx[i];
        sum=b[ip];
        b[ip]=b[i];
        if (ii)
            for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
```

```
            else if (sum) ii=i;
            b[i]=sum;
        }
        for (i=n;i>=1;i--) {
            sum=b[i];
            for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
            b[i]=sum/a[i][i];
        }
}

double sort2(n,ra.rb)
int n;
double ra[],rb[];
{
    int l,j,ir,i;
    double rrb,rra;

    l=(n >> 1)+1;
    ir=n;
    for (;;) {
        if (l > 1) {
            rra=ra[--l];
            rrb=rb[l];
        } else {
            rra=ra[ir];
            rrb=rb[ir];
            ra[ir]=ra[1];
            rb[ir]=rb[1];
            if (--ir == 1) {
                ra[1]=rra;
                rb[1]=rrb;
                return;
            }
        }
        i=l;
        j=l << 1;
        while (j <= ir) {
            if (j < ir && ra[j] < ra[j+1]) ++j;
            if (rra < ra[j]) {
                ra[i]=ra[j];
                rb[i]=rb[j];
                j += (i=j);
```

```
            }
            else j=ir+1;
        }
        ra[i]=rra;
        rb[i]=rrb;
    }
}

void nrerror(error_text)
char error_text[];
{
    void exit();

    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

int *ivector(nl,nh)
int nl,nh;
{
    int *v;

    v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl;
}

float *vector(nl,nh)
int nl,nh;
{
    float *v;

    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
```

```
    double *v;

    v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl;
}

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    float **m;

    m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
    int i;
    double **m;

    m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
    if (!m) nrerror("allocation failure 1 in dmatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
        if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
        m[i] -= ncl;
    }
    return m;
}
```

```
void free_ivector(v,nl,nh)
int *v,nl,nh;
{
    free((char*) (v+nl));
}

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
    free((char*) (v+nl));
}

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}
```

# *Bibliography*


1. Apostol, Tom M. *Mathematical Analysis* (second Edition). Reading, Massachusetts: Addison-Wesley, 1974.
2. Barnsley, Michael. *Fractals Everywhere*. San Diego: Academic Press, 1988.
3. Burns, Thomas J. *A Non-homogeneous, Spatio-Temporal Wavelet Multiresolution Analysis and Its Application to the Analysis of Motion*. PhD dissertation, Air Force Institute of Technology, 1993.
4. Casdagli, Martin. "Chaos and Deterministic versus Stochastic Non-linear Modelling," *Journal of the Royal Statistical Society B, 54*(2):303–328 (1991).
5. Casdagli, Martin C. and Andreas S. Weigend. "Exploring the Continuum Between Deterministic and Stochastic Modeling." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 347–366, Addison-Wesley, 1994.
6. Devaney, Robert L. *An Introduction to Chaotic Dynamical Systems* (second Edition). Redwood City, California: Addison-Wesley, 1989.
7. Eckmann, J.P. and D. Ruelle. "Ergodic theory of chaos and strange attractors," *Reviews of Modern Physics, 57*(3):617–656 (1985).
8. Farmer, J. Doyne and John J. Sidorowich. "Predicting Chaotic Time Series," *Physical Review Letters, 59*(8):845–848 (1987).
9. Fielding, K. H. and D. W. Ruck. "Automatic Object Recognition Using Image Sequences," *IEE Journal on Vision, Image, and Signal Processing* (1994). Submitted for publication.
10. Fielding, K. H., et al. "Spatio-Temporal Pattern Recogniton Using Hidden Markov Models." *Proceedings of the SPIE Vol. 2032 Neural and Stochastic Methods in Image and Signal Processing II*. 144–154. July 1993.
11. Fielding, K. H., et al. "Spatio-Temporal Pattern Recogniton Using Hidden Markov Models," *IEEE Transactions on Aerospace and Electronic Systems* (1994). Submitted for publication.
12. Fielding, Kenneth H. *Spatial-Temporal Pattern Recognition Using Hidden Markov Models*. PhD dissertation, Air Force Institute of Technology, 1994.
13. Fraser, Andrew M. and Harry L. Swinney. "Independent coordinates for strange attractors from mutual information," *Physical Review A, 33*(2):1134–1140 (1986).
14. Fuller, Wayne A. *Introduction to Statistical Time Series*. New York: John Wiley and Sons, 1976.
15. Glass, Leon and Michael C. Mackey. *From Clocks to Chaos: The Rhythms of Life*. Princeton, New Jersey: Princeton University Press, 1988.

16. Gleick, James. *Chaos: Making a New Science*. New York: Viking Penguin, 1988.

17. Grassberger, Peter and Itamar Procaccia. "Measuring the Strangeness of Strange Attractors," *Physica D, 9*:189–208 (1983).

18. Hirsch, Morris W. and Stephen Smale. *Differential Equations, Dynamical Systems, and Linear Algebra*. San Diego: Academic Press, 1974.

19. Lang, Serge. *Analysis I*. Reading, Massachusetts: Addison-Wesley, 1968.

20. Libby, Edmund W. *Application of Sequence Comparison Methods to Multisensor Data Fusion and Target Detection*. PhD dissertation, Air Force Institute of Technology, 1993.

21. Lorenz, E. N. "Deterministic Nonperiodic Flow," *Journal of the Atmospheric Sciences, 20*:130–141 (1963).

22. Massopust, Peter R. "Vector-valued fractal interpolation functions and their box dimension," *Aequationes Mathematicae, 42*:1–22 (1991).

23. Mazel, David S. and Monson H. Hayes. "Using Iterated Function Systems to Model Discrete Sequences," *IEEE Transactions on Signal Processing, 40*(7):1724–1734 (1992).

24. Montgomery, Douglas C. and others. *Forecasting and Time Series Analysis* (second Edition). New York: McGraw-Hill, 1990.

25. Moon, Francis C. *Chaotic and Fractal Dynamics: An Introduction for Applied Scientists and Engineers*. New York: John Wiley and Sons, 1992.

26. Mozer, Michael C. "Neural Net Architectures for Temporal Sequence Processing." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 243–264, Addison-Wesley, 1994.

27. Pineda, Fernando J. and John C. Sommerer. "Time Series Prediction by Using Delay Coordinate Embedding." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 367–385, Addison-Wesley, 1994.

28. Press, William H. and others. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, UK: Cambridge University Press, 1988.

29. Press, William H. and others. *Numerical Recipes in C: The Art of Scientific Computing* (second Edition). Cambridge, UK: Cambridge University Press, 1992.

30. Rabiner, Lawrence R. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE77*. 257–286. 1989.

31. Rietman, Edward. *Exploring the Geometry of Nature: Computer Modeling of Chaos, Fractals, Cellular Automata and Neural Networks*. Blue Ridge Summit, Pennsylvania: Windcrest Books, 1989.

32. Sauer, Tim. "Time Series Prediction by Using Delay Coordinate Embedding." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 175–193, Addison-Wesley, 1994.

33. Sauer, Tim and others. *Embedology*. Technical Report 91-01-008, Santa Fe Institute, 1991.

34. Seibert, Michael and Allen M. Waxman. "Adaptive 3-D Object Recognition from Multiple Views," *IEEE Transactions on Pattern Analysis and Machine Intelligence, 14*(2):107–124 (1992).

35. Seydel, Rudiger. *From Equilibrium to Chaos: Practical Bifurcation and Stability Analysis*. New York: Elsevier Science, 1988.

36. Stright, James R. *A Neural Network Implementation of Chaotic Time Series Prediction.* MS thesis, Air Force Institute of Technology, 1988.

37. Takens, Floris. "Detecting strange attractors in turbulence." *Dynamical Systems and Turbulence, Warwick 1980*, Lecture Notes in Mathematics 898, edited by D. A. Rand and L. S. Young. 366–381. Springer-Verlag, 1981. Printed in Germany.

38. Theiler, J., et al. "Detecting Nonlinearity in Data with Long Coherence Times." *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld, 429–455, Addison-Wesley, 1994.

39. Theiler, James. "Estimating fractal dimension," *Journal of the Optical Society of America A*, 7(6):1055–1073 (1990).

40. Weigend, Andreas and Neil Gershenfeld. "Time Series Prediction." Notes provided at a tutorial at NIPS*93, Denver, CO, November 1993.

41. Weigend, Andreas S. and Neil A. Gershenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Reading, Massachusetts: Addison-Wesley, 1994.

42. Wolf, Alan and others. "Determining Lyapunov Exponents from a Time Series," *Physica 16D*, 285–317 (1985).

## *Vita*

Capt James R. Stright grew up in northwestern Pennsylvania. He graduated from high school in Meadville, Pennsylvania, and from Allegheny College in Meadville, where he majored in mathematics. He joined the Air Force in 1983, and earned a bachelor's degree in electrical engineering from Gannon University in Erie, Pennsylvania in 1984. Following graduation from OTS in August 1984, his first assignment was to the National Computer Security Center at Fort Meade, Maryland, where he served as the government team leader for development of an encryption device. In 1988 he received an M.S.E.E. degree from the Air Force Institute of Technology (AFIT). His next assignment was to Aeronautical Systems Division, Wright-Patterson Air Force Base, where he served as a computer security systems engineer on the Advanced Tactical Fighter program. Capt Stright returned to AFIT in 1990 to pursue a Ph.D. in electrical engineering. His doctoral research focuses on the prediction of chaotic time series. He is a member of SIAM.

Permanent address: 3049 Blue Green Drive
Beavercreek, OH 45431

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | June 1994 | *Doctoral Dissertation* |

**4. TITLE AND SUBTITLE**
Embedded Chaotic Time Series: Applications in Prediction and Spatio-temporal Classification

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
James R. Stright, Capt, USAF

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Air Force Institute of Technology, WPAFB OH 45433-6583

**8. PERFORMING ORGANIZATION REPORT NUMBER**
AFIT/DS/ENG/94J-04

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
WL/MN (Samuel Lambert)
101 West Eglin Blvd., Suite 130
Eglin AFB, FL 32542-6811

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Approved for Public Release; Distribution Unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

The Deterministic Versus Stochastic algorithm developed by Martin Casdagli is modified to produce two new prediction methodologies, each of which selectively uses embedding space nearest neighbors. Neighbors which are considered prediction-relevant are retained for local linear prediction, while those which are considered likely to represent noise are ignored. For many time series, it is shown possible to improve on local linear prediction with both of the new algorithms. Furthermore, the theory of embedology is applied to determine a length of test sequence sufficient for accurate classification of moving objects. Sequentially recorded feature vectors of a moving object form a training trajectory in feature space. Each of the sequences of feature vector components is a time series, and under certain conditions, each of these time series has approximately the same fractal dimension. The embedding theorem is applied to this fractal dimension to establish a number of observations sufficient to determine the feature space trajectory of the object. It is argued that this number is a reasonable test sequence length for use in object classification. Experiments with data corresponding to five miiitary vehicles (observed following a projected Lorenz trajectory on a viewing sphere) show that this number is indeed adequate.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Time Series Prediction, Embedology, Motion Analysis | 147 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |